TRABAJO FINAL DE CARRERA

TÍTULO: Asignación de Canales en redes ad-hoc 802.11 multi-radio

TITULACIÓN: Ingeniería Técnica de Telecomunicaciones, especialidad Telemática

AUTOR: Manuel José Coronado Francisco

DIRECTOR: Eduard Garcia Villegas

FECHA: 17 de Julio de 2006

Título: Asignación de canales en redes ad-hoc 802.11 multi-radio

Autor: Manuel José Coronado Francisco

Director: Eduad García Villegas

Fecha: 17 de Julio de 2006

Resumen

Hasta hace poco tiempo, para tener acceso a Internet nos encontrábamos con la necesidad de estar conectados físicamente a una red, teniendo la necesidad de estar buscando una toma de red y un cable para conectar nuestro ordenador. En estos últimos años, la gran necesidad de estar conectado a Internet, la movilidad de los usuarios y la disposición de elementos inalámbricos a unos precios asequibles han motivado de forma masiva la creación de redes de área local inalámbricas, tanto en oficinas y empresas, como en hogares, hoteles, bibliotecas, etc. Estas redes están evolucionando de manera vertiginosa, mejorando sus prestaciones gracias a nuevos tipos de redes con configuraciones físicas y lógicas distintas. Un ejemplo de estas son las redes malladas inalámbricas (mesh). Las redes mesh se basan en un funcionamiento ad-hoc y la ausencia de infraestructura (puntos de acceso fijos), permitiendo topologías más dinámicas y distribuidas tal y como se prevé en las actuales arquitecturas propuestas para la futura 4G.

Este tipo de redes funcionan principalmente sobre bandas de frecuencias de libre uso. Dichas bandas están muy limitadas: según los estándares de telecomunicaciones europeos (ETSI) podrían utilizarse 13 canales, pero debido al ancho de banda que utiliza el estándar de la IEEE 802.11 (estándar de facto para la implementación de redes de área local inalámbricas, WLAN), únicamente se pueden usar 3 canales no solapados. El uso de más canales provocaría interferencias, bajando la calidad del canal, teniendo como consecuencia unas velocidades de transmisión bajas. En entornos urbanos, donde el crecimiento ha sido masivo, empiezan a aparecer los primeros problemas de coexistencia entre las diferentes redes inalámbricas, puesto que al usar una misma banda de frecuencia sin control, nos encontramos con que las interferencias provocan una importante pérdida de eficiencia. Se hace necesario por tanto, realizar una gestión inteligente de los escasos recursos radio con el fin de mejorar la calidad que percibe el usuario.

Con este proyecto lo que se pretende es evitar este problema y hacer un buen uso del espectro frecuencial para así poder trabajar con una calidad del canal óptima minimizando las interferencias y poder sacar de esta forma el máximo rendimiento a nuestras redes. Para lograrlo, se utilizarán dispositivos con dos antenas. Esto nos permitirá agrupar nodos en clusters de manera que con una antena se realizan tareas de control y comunicaciones inter–cluster, y con la otra antena se permite la comunicación entre miembros del mismo cluster.

Title: Assignment of channels in networks ad-hoc 802.11 multi-radio

Author: Manuel José Coronado Francisco

Director: Eduard García Villegas

Date: July, 17th 2006

Overview

Not so long ago, in order to have Internet access we had to be physically connected to a network, we had to search a network socket and a cable to plug our computer. In the recent years, the need to be connected to Internet, the users' mobility and the availability of wireless devices at affordable prices have motivated the creation of wireless local area networks, not only in offices and companies, but also in homes, hotels, libraries, etc. These networks are rapidly evolving, improving their performance thanks to new types of physical configurations and different logics. Examples of these are the wireless mesh networks. The mesh networks are based on an ad-hoc nature and the absence of infrastructure (fixed access points), allowing more dynamic and distributed topologies as it is foreseen in the current architectures proposed for the future 4G.

This type of networks use unlicensed frequency bands. The mentioned bands are very limited: according to the European Standards of Telecommunications (ETSI) there are 13 channels defined, but due to the bandwidth that is used by the IEEE 802.11 standard (standard de facto for the implementation of wireless local area networks, WLAN) signals, only 3 are not overlapping. The use of more channels would produce interference, lowering the quality of the channel and decreasing the connection speed. In urban environments, where the growth of these networks has been massive, the first coexistence problems appear in the different wireless networks, because they use the same frequency band without control, so the main problem of this type of networks is met, interference, that causes transmission errors, collisions and the consequent loss of efficiency. It becomes necessary, to carry out an intelligent management of the scanty resources in order to improve the quality that the user perceives.

With this project what is claimed is to avoid this problem and to do a good use of the frequency spectrum so that we can achieve an ideal channel quality by minimizing interference. Hence, we are able to obtain the best performance in our networks. To achieve it, we use devices with two antennas (so called "cube") one of which will allow us to realize control tasks and inter – cluster communications, and with the second antenna we'll be able to allow communications between members of the cluster.

Índice

INTRODUCCIÓN

Hasta hace poco tiempo, para tener acceso a Internet nos encontrábamos con la necesidad de estar conectados físicamente a una red, teniendo la necesidad de estar buscando una toma de red y un cable para conectar nuestro ordenador.

En estos últimos años, la gran necesidad de estar conectado a Internet, la movilidad de los usuarios y la disposición de elementos inalámbricos a unos precios asequibles han motivado de forma masiva la creación de redes de área local inalámbricas, tanto en oficinas y empresas, como en hogares, hoteles, bibliotecas, etc.

Como el resto de tecnologías de la información, las redes inalámbricas están en constante evolución. Las WLANs más comunes están basadas en el uso de puntos de acceso (APs) que forman parte de una infraestructura fija, pero con el inconveniente de que todos los usuarios que se quieran conectar a la red han de estar dentro del área de cobertura del AP. A diferencia de ésta topología de redes, nos encontramos también con las redes ad-hoc. Con estas se crean principalmente enlaces punto a punto, cuya única particularidad es la coincidencia en el ESSID (nombre que se le da a la red), igual para todos los nodos de la red.

La finalidad de una topología Mesh Wireless (Mallada Inalámbrica) es mezclar estas dos arquitecturas, dando una mayor movilidad a los usuarios y permitiendo una nueva topología totalmente dinámica al seguir una filosofía distribuida donde los nodos son terminales cliente y además como parte activa de la red, son también responsables de distribuir y encaminar datos de otros terminales cercanos. Por ejemplo, en una red de infraestructura fija todos los nodos han de estar dentro del área de cobertura del punto de acceso (AP), pues ahora bien si hubiesen dispositivos “Mesh”, dispositivos que no estuviesen dentro del área de cobertura del AP pero si de un dispositivo “Mesh”, estarían de forma indirecta perteneciendo al área de cobertura del AP, pudiendo crear de esta forma un acceso a redes con infraestructura fija mediante cadenas de enlaces o lo que es lo mismo, mediante un acceso multi-salto.

Este tipo de redes funcionan sobre bandas de frecuencias de libre uso. Dichas bandas están muy limitadas: según los estándares de telecomunicaciones europeos (ETSI) podrían utilizarse 13 canales, pero debido al ancho de banda que utiliza el estándar de la IEEE 802.11 (estándar de facto para la implementación de redes de área local inalámbricas, WLAN), únicamente se pueden usar 3 canales no solapados. El uso de más canales provocaría interferencias, bajando la calidad del canal, teniendo como consecuencia unas velocidades de transmisión bajas.

En entornos urbanos, donde el crecimiento ha sido masivo, empiezan a aparecer los primeros problemas de coexistencia entre las diferentes redes inalámbricas, puesto que al usar una misma banda de frecuencia sin control, nos encontramos con el principal problema de este tipo de redes, las interferencias, provocando en las transmisiones errores, colisiones y la consiguiente pérdida de eficiencia. Se hace necesario por tanto, realizar una gestión inteligente de los escasos recursos radio con el fin de mejorar la calidad que percibe el usuario.

Con este proyecto lo que se pretende es evitar este problema y hacer un buen uso del espectro frecuencial para así poder trabajar con una calidad del canal óptima minimizando las interferencias y poder sacar de esta forma el máximo rendimiento a nuestras redes. Para lograrlo, se utilizaran dispositivos con dos antenas ya que, con una sola interfaz, todos los nodos deben estar sintonizados en el mismo canal para poder transmitir, causando numerosos problemas de interferencias y colisiones. En cambio, con dos interfaces se pueden agrupar los nodos de una red ad-hoc en clusters de manera que se usa una interfaz en un canal para comunicarse con los elementos de un cluster, y la otra para la comunicación con miembros de clusters vecinos.

En el Capítulo 1 del presente proyecto, se ha realizado un estudio de diferentes algoritmos destinados a la agrupación de dispositivos de forma automática (clustering) y a la asignación de canales de forma eficiente para todos los grupos de dispositivos formados, viendo así diferentes formas de crear clusters, comparándolos posteriormente en su complejidad y en la eficiencia que proporcionan.

En el Capítulo 2, tras realizar el estudio se ha escogido uno de los algoritmos de clustering implementando las mejoras oportunas para un mejor funcionamiento. Dentro de este mismo capítulo también se implementa el algoritmo de asignación de canales (cuyo algoritmo se ejecuta tras finalizar el algoritmo de clustering), que al igual que anteriormente, se partirá de los estudios realizados en el Capítulo 1 para rediseñar un algoritmo que cumpla nuestras necesidades.

En el Capítulo 3, una vez diseñados y programados los algoritmos, se procede a detallar la implementación desarrollada, explicando detalladamente el funcionamiento de los mismos, describiendo todos los casos posibles con los que nos podemos encontrar en la ejecución de dicho algoritmo.

En el Capítulo 4, se han realizado pruebas de rendimiento de la red en tres escenarios con características distintas. Para cada uno de ellos, se han realizado caracterizaciones de retardo y throughput, comparando los casos en los que se asigna un único canal para todos los nodos, con el caso de nodos con dos interfaces.

Para finalizar, se concluye en el Capítulo 5 con una revisión de los aspectos más destacables de este proyecto e indicando también tanto posibles líneas futuras, como el impacto medio ambiental que tendría nuestra implementación.

CAPÍTULO 1: ALGORITMOS

1.1. Introducción.

En el presente capítulo nos proponemos realizar un estudio y a analizar diferentes propuestas de algoritmos que son aptos para el desarrollo de una infraestructura Mesh Wireless.

Los algoritmos a tratar en cuestión en este capítulo son: algoritmo HCC; algoritmo Clustering; algoritmo MIX; algoritmo DCA. Estos algoritmos están divididos en el presente documento en dos secciones, las cuales van en función a las funciones que desarrollan cada uno de los algoritmos.

Palabras tales como cluster, clusterhead y nodo vecino, nos las encontraremos repetidas a lo largo del documento, las cuales explicaremos a continuación para que no haya confusiones.

- Cluster: un cluster es un conjunto de nodos interconectados entre sí (para este caso, mediante enlaces radio) que comparten un medio y unas características para la transmisión y recepción de datos, comunes.

- Clusterhead: se entiende por clusterhead, al nodo que tomará decisiones referente al canal a utilizar en el cluster al que pertenece (en función al algoritmo utilizado).

- Nodo vecino: entendemos como nodo vecino, todo aquel nodo que está como máximo a un salto de un mismo nodo.

Dadas estas definiciones básicas, en los próximos apartados se da paso a la explicación de los algoritmos.

1.2. Algoritmos de Clustering.

En este apartado estudiaremos dos tipos de algoritmos: el algoritmo HCC (Highest Connectivity Cluster) y el algoritmo de Clustering presentado en [3]. La finalidad de ambos algoritmos es la misma, dividir los elementos de una red mesh inalámbrica realizando varios clusters, a los cuales después se les asignaran unos canales u otros en función de unos parámetros que estudiaremos en el punto siguiente.

1.2.1. Algoritmo HCC (Highest Connectivity Cluster).

Tal y como se puede observar en el artículo [1], el algoritmo HCC está basado en las siguientes reglas:

- Un nodo que no ha elegido su clusterhead, aún será un nodo "descubierto", de otra manera, este sería un nodo "asociado" (al cluster).

- Un nodo es elegido clusterhead si este nodo es el que tiene un número mayor de nodos vecinos "descubiertos" conectados.

- Aquel que haya sido elegido, realizará el papel de clusterhead.

Para dicho algoritmo partimos de la base en la que todos los nodos conocen sus nodos vecinos, tanto aquellos que ya están asociados a un cluster (nodo asociado), como a los que no están asociados a ningún cluster (nodo descubierto). Para ello los nodos se transmiten una trama CMT [1] (Clustered Multi-channel two-radio) en la que figuran tres campos, tal y como se puede observar en la figura siguiente (**Fig.1.1**).

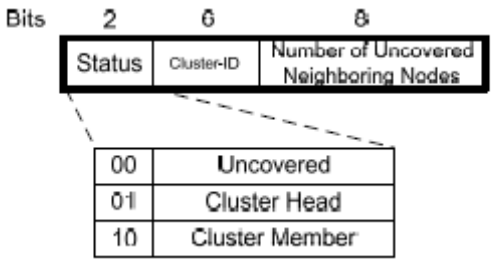

Fig.1.1: Trama CMT

Con estos tres campos es suficiente para informar a los nodos vecinos referente el estado en el que se encuentra el nodo (campo Status), el identificador del cluster al que pertenece (Cluster-ID) y del número de nodos vecinos descubiertos que hay alrededor de un mismo nodo. Este último campo será el que se utilizará después para asignar a un nodo la función de clusterhead.

Como ejemplo de dicho algoritmo podemos observar la siguiente figura (**Fig.1.2**), en la que hay 10 nodos, dónde los nodos 8, 5 y 7 son clusterheads; y los nodos 10, 9, 3 y 2 son gateways. Como se puede observar en la misma, el área de cobertura vendría dado por el círculo que engloba los nodos 10, 2, 8, 1, 6, 3 y 9; y los enlaces establecidos entre los nodos vienen representados con la unión de los mismos mediante líneas.

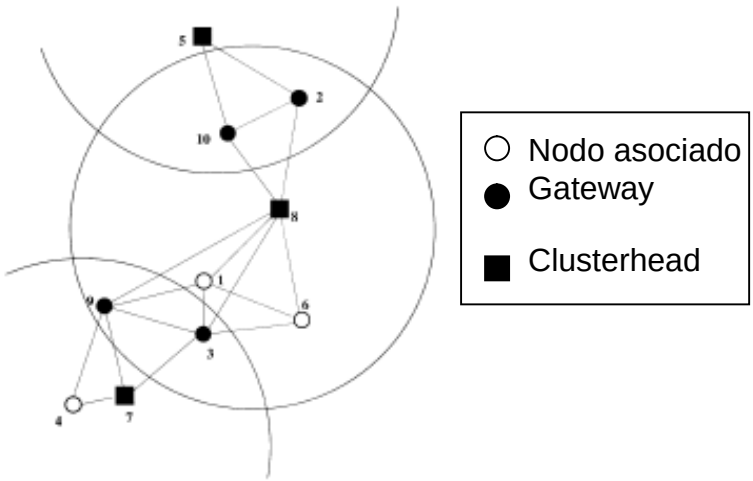

Fig.1.2: Ejemplo algoritmo HCC.

Para analizar la figura anterior (**Fig.1.2**) realizaremos una serie de suposiciones lógicas en referencia a un primer estado inicial, la cuales se citan a continuación:

- Todos los nodos de la red están en estado “descubierto”, es decir, no están asociados a ningún cluster (puesto que teóricamente aún no se ha creado).

- Todos los nodos conocen los nodos que tienen a su alrededor (nodos a un salto).

Una vez dicho esto, pasamos a la ejecución del algoritmo. Podemos deducir a simple vista que el nodo 8 (nodo más céntrico), será el que se proclame clusterhead, lo cual será así, puesto que, tal y como hemos comentado anteriormente, será el nodo que tiene un mayor número de nodos vecinos descubiertos a un salto a su alrededor, asignándose así la función de clusterhead.

También comentar la existencia de una variante del HCC, conocido como Lowest-ID [1], cuyo funcionamiento se comentará a continuación sin entrar mucho en detalle.
Con el algoritmo Lowest-ID, periódicamente cada nodo difunde en broadcast una lista de nodos en la que figuran los nodos que éste puede escuchar, es decir, que están a un salto de él, incluyéndose así mismos. A continuación, basándonos en las siguientes reglas obtendremos los clusters bien definidos:

- Un nodo que solo escuche ID más altos que el suyo, ese nodo será clusterhead, pasando a realizar funciones como tal.

- Los nodos que pueden escuchar dos o más clusterheads, serán nodos gateways.
- El resto de nodos, serán nodos ordinarios.

1.2.2. Algoritmo de Clustering distribuido.

Como hemos mencionado anteriormente, el presente algoritmo consiste en partir una red en varios clusters. Una medida óptima de cluster vendría dada por la compensación entre el rehúso espacial, un retardo mínimo, y también en función de la potencia de transmisión. Para el algoritmo de clustering distribuido descrito en [3], asumiremos que la potencia de transmisión es fija y uniforme para todos los nodos de la red.

En todos los clusters, los nodos se pueden comunicar con todos los otros nodos vecinos, hasta como mucho dos saltos. Los clusters se construyen a partir de un identificador del nodo (ID). Para dicho algoritmo se proponen las siguientes suposiciones, las cuales son comunes para la gran mayoría de protocolos de enlaces de transmisión radio:

- Cada nodo tiene un único ID y conoce los ID's de sus vecinos a un salto.
- Un mensaje enviado por un nodo es recibido correctamente dentro de un tiempo finito por todos los nodos vecinos a un salto.
- La topología de la red no cambiará durante la ejecución del algoritmo.

El algoritmo de clustering distribuido converge muy rápidamente. Lo pasamos a explicar a continuación al mismo tiempo que lo comentamos con un ejemplo. Consideremos la siguiente topología (**Fig.1.3**).

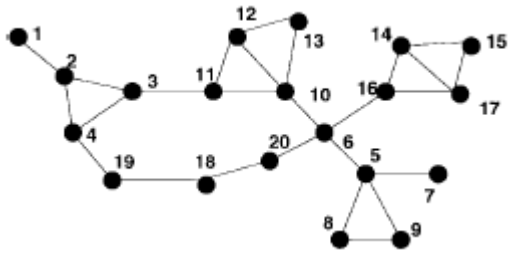

Fig.1.3: Topología de la red

Todos los nodos se intercambian entre si cierta información, en la que dan a conocer cual es su ID y su clusterhead (dicha información se transmite cada cierto tiempo, predefinido). En este algoritmo, el clusterhead, será aquel nodo cuya ID sea la más baja dentro de un mismo cluster.

Cuando un nodo se activa por primera vez, éste al no conocer ningún ID mejor que el suyo (puesto que aún no conoce ningún vecino), se considerará

clusterhead, y empezará a transmitir su ID y su clusterhead (que será el mismo que su ID).

Siguiendo el diagrama de estados que se puede observar en la siguiente figura (**Fig.1.4**), en el momento en que ya conoces los vecinos que te rodean, cada uno de los nodos mirarán los paquetes recibidos de los nodos vecinos, y pasarán a comprobar que el campo ID es igual que el campo clusterhead, para así saber si dicho nodo es clusterhead. Si esta comparación se observa que no son iguales, simplemente almacenamos en una tabla, la ID del nodo y el clusterhead al que pertenece. Pero ahora bien, si en ésta comparación ambos campos son iguales, pasamos a comparar la ID propia con la ID que ha recibido de su vecino. Si el ID del nodo vecino es menor que su ID, pasaremos a realizar otra comparación con el ID del nodo vecino y el ID del clusterhead al que estamos conectado (pudiendo ser el mismo nodo). Con ésta última comparación, si el ID del nodo vecino, ha sido inferior, éste pasará a ser clusterhead de dicho nodo, formando (si no esta formado ya anteriormente) un nuevo cluster. Veamos nodos 1 y 2, el nodo 1 conoce los datos del nodo 2 y viceversa, además que el nodo 2 tiene más nodos conectados. El nodo 2 observará que el nodo1 en la información que le ha pasado, los campos (ID y clusterhead) son iguales (de un principio se ha considerado ya clusterhead) y que además tiene un ID más bajo que el suyo. De esta forma, el nodo 2 se asociará al nodo 1 formando un cluster y la información que difundirá a partir de ahora el nodo 2 en el campo clusterhead, será que pertenece al nodo 1.

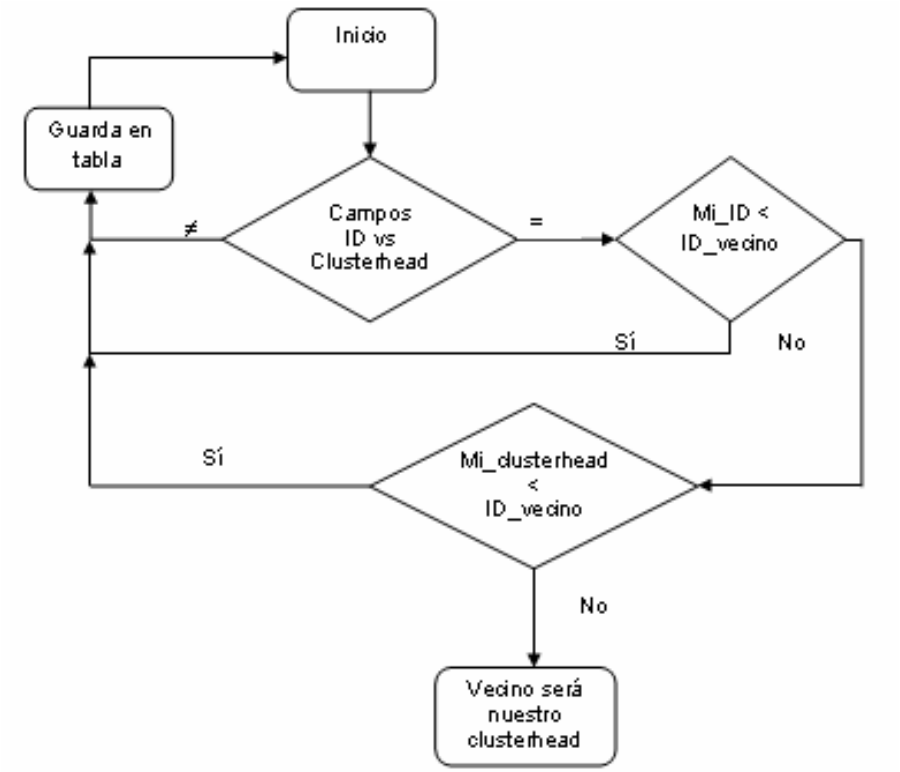

Fig.1.4: Diagrama de estado

Siguiendo dicho diagrama, para todos y cada uno de los nodos que hemos dibujado en la topología anterior (**Fig.1.3**), llegaríamos al punto en que se han realizado todos los clusters, agrupándose finalmente tal y como se muestra a continuación (**Fig.1.5**).

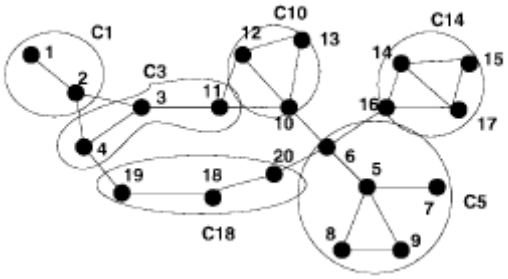

Fig.1.5: Clustering

Para un correcto funcionamiento del algoritmo, cada nodo eventualmente determina su cluster, es decir, ejecuta el algoritmo para comprobar si ha habido cambios en el cluster. También con este algoritmo, se puede observar que como máximo, dos nodos cualesquiera pertenecientes a un mismo cluster están como máximo a una distancia de dos saltos.

Para finalizar, simplemente comentar que cada uno de los nodos transmiten un único mensaje durante la ejecución del algoritmo.

1.3. Algoritmos de asignación de canales.

Tras haber visto el funcionamiento de dos algoritmos totalmente diferentes para la realización del clustering, ahora hace falta ver como se asignan los canales en los diferentes clusters que se han formado, con la finalidad de hacer un uso eficiente del espectro radioeléctrico. La reutilización espacial en tales redes puede permitir a múltiples comunicaciones proceder simultáneamente, de ahí proporcionalmente mejorar el rendimiento de red total. En este apartado explicaremos dos algoritmos de asignación de canales, el primero, llamado algoritmo MIX (Minimum Interference Channel selection), y un segundo llamado algoritmo DCA (Distributed Channel Allocation).

1.3.1. Algoritmo MIX (Minimum Interference Channel selection).

El presente algoritmo esta basado en la minimización de la interferencia co-canal entre los clusters trabajando con un mínimo de potencia de transmisión en las comunicaciones intra-clusters.

Este algoritmo se explica en [1], donde se utilizan APs con dos interfaces radio. Estas dos interfaces radio son configuradas de manera que, a una interfaz radio (llamada interfaz primaria) se le asigna por defecto un canal pre-configurado fijo, cuya finalidad será realizar todas las comunicaciones inter-clusters, mientras que las comunicaciones intra-clusters se realizarán con la interfaz radio secundaria usando un canal escogido por el clusterhead, el cual aplicará el algoritmo MIX (selección de canal de interferencia mínima) obteniendo de esta manera el canal que menos interferencia co-canal tiene.

La elección del clusterhead para este algoritmo, tal y como se explica en el artículo [4], se realiza usando el algoritmo HCC (explicado en **1.2.1**). Así pues, una vez el algoritmo HCC ha escogido el clusterhead, y ha formado los diferentes clusters, se pasará a la asignación de canales usando el algoritmo MIX.

1.3.2. Algoritmo DCA (Distributed Channel Allocation).

En esta sección pasaremos a describir el funcionamiento del algoritmo DCA presentado en [4]. Cabe comentar que dicho algoritmo está basado en el esquema de Clustering presentado en [3], con el cual, a partir de los pesos (prioridad) de cada uno de los nodos, obtendremos un Clusterhead (CH). Este CH, conocerá todos los pesos de los CHs y todos los nodos que están a una distancia de dos saltos como máximo del mismo.

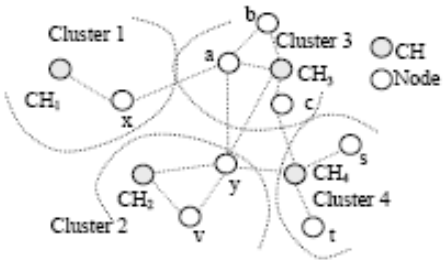

Fig.1.6: Topología de la red

Para la explicación de dicho algoritmo, tomaremos como ejemplo la anterior figura (**Fig.1.6**), para simplificar la explicación y que el lector pueda entender mediante ejemplos dicho algoritmo.

Un concepto que debe quedar claro es el del Conjunto de Vecinos de un cluster (NS en sus siglas en inglés): NS_C es el conjunto de nodos que están como mucho a dos saltos del cluster *C* y que además, no pertenecen a *C*. Por ejemplo, según **Fig.1.6**, NS_1 = {*a*, *b*, *CH3*, *y*}. A partir de ahora se trata de asignar canales a los clusters de manera que no haya conflictos con ninguno de los nodos de su NS.
Los Nodos CH proceden a colorear su cluster sólo cuando sus vecinos CH de mayor peso (prioridad) ya han coloreado completamente su cluster (i.e. ya han seleccionado un canal para los miembros de su cluster).

Uno de los problemas que nos podemos encontrar a la hora de colorear es el del cluster oculto. Por ejemplo *x* e *y* se enlazan a partir de *a* (cada uno en un cluster diferente). Para combatir este problema habrá que añadir una tarea para que no haya conflictos con la asignación de colores. Es por ello por lo que es necesario que la información de los colores que se usan en el cluster corra también a través de sus fronteras.

DCA comienza en el punto donde cada CH conoce cual es su peso W, y también conoce el de sus vecinos.

Ejemplo: CH_2 conoce $W(CH_1)$, $W(CH_3)$, $W(CH_4)$

Un CH procede a colorear su cluster solo cuando sus vecinos de mayor peso ya han coloreado completamente su cluster.

Ejemplo: Si $W(CH_2) > W(CH_3) > W(CH_4) > W(CH_1)$
Se comenzara a colorear por CH_2, a continuación por CH_3, y así sucesivamente.

En este caso, una vez completada la asignación de colores de CH_2 y CH_3, esta se propaga, y así CH_1 y CH_4 empezaran a colorear simultáneamente. Cabe destacar que no es posible que haya conflictos de colores entre los clusters 1 y 4, ya que sus NS no contienen nodos del otro cluster.

Cada nodo en cualquier cluster mantiene una tabla de colores (CT) conteniendo una lista de sus nodos vecinos que pertenecen a un CH que tiene un W más alto que el del CH al que pertenecen.

Ejemplo: $W(CH_2) > W(CH_3)$
CT_a= {v, y, CH_2} y CT_y = {∅}

Cuando un nodo recibe la información de los colores referente a los nodos vecinos, la lista se actualiza, y se prohíben todos estos colores en ese mismo nodo. Además el CH tiene una tabla llamada *Cluster Color Table* (CCT). En esta CCT figura un registro para cada nodo de su cluster y los colores prohibidos en cada uno. Un CH comienza a colorear sólo cuando este CCT está completo, es decir se conocen todos los colores prohibidos para cada nodo que están en el cluster.

Con DCA circulan muchos mensajes, de los cuales se definen 3 tipos:

- CA (Chanel Assignament message) **à** enviado por el CH.
- UP (Update Message) **à** apuntado a vecinos de 1 salto para que sean conscientes de los colores prohibidos.
- INF (Information Message) **à** alivia el problema de cluster oculto notificando el CH de un color prohibido para un nodo de su cluster.

- Explicación detallada de los tres tipos de mensajes.

- CA: Consiste en un par ordenado del tipo (Nodo ID, Color Asignado) para cada nodo del cluster. Un nodo primero identifica si el mensaje ha sido enviado por su CH. Entonces éste extrae el color que se le ha asignado tanto a él, como a los nodos a un salto que son miembros de su mismo cluster. Así el nodo *y* identifica el color asignado a él, a CH_2, *v*. *y* entonces difunde un paquete BroadCast de subida (UP) conteniendo esta información.

- UP: Los mensajes de actualización van dirigidos a todos los nodos a un salto de un nodo recientemente coloreado, notificando los colores asignados a su cluster.

  Ejemplo: Cuando el nodo *a* recibe un mensaje UP desde *y*, la CT_a es actualizada con los colores asignados a *y*, *v*, y CH_2. Un mensaje UP es difundido con un INF BroadCast, llegando así a todos los nodos a un salto del nodo. Los nodos pertenecientes al cluster de *y* no hacen caso de este mensaje.

  Lo que haría también CH_3 al recibir este mensaje (de un nodo no perteneciente al propio Cluster) añade el color del nodo emisor como un color prohibido para todos los nodos de su cluster.

- INF: Existen dos tipos de mensajes.
  - INF BroadCast (IB) **à** mensaje generado por un nodo resultado de un mensaje UP. Es útil para su propio CH y todos los otros vecinos.
  - INF UniCast (IU) **à** mensaje dirigido al CH, es generado cuando un nodo recibe un IB desde un remitente que no pertenece a su Cluster.

  Ejemplo: Un mensaje IB enviado por *a* es usado por CH_3 para actualizar su CCT. Un nodo del cluster adyacente, *x* escucha este mensaje, extrae su lista de 1 salto y actualiza su CT. Un mensaje IU lo mandará *x* hacia CH_1 solventando así el problema de cluster oculto entre 1 y 2.

1.4. Análisis de los algoritmos.

A continuación se pasará a describir los resultados de los análisis que se realizaron de los algoritmos anteriormente presentados [1][2][3][4] para así, posteriormente, llegar a una conclusión final.

Antes de pasar a explicar las pruebas analíticas que se realizaron, cabe comentar una observación en referencia a la potencia de transmisión, ya que ésta determinará la topología de la red, teniendo un impacto directo en la formación del cluster. Para el caso en que tengamos una potencia de transmisión alta, el numero medio de saltos entre origen-destino será un valor bajo, puesto que los clusters generados tendrán un mayor alcance, teniendo como consecuencia un mayor número de nodos asociados. Como contra partida a transmitir con una potencia elevada, nos encontramos con que el nivel de interferencia es muy elevado. Por otra parte, podríamos transmitir con una potencia de transmisión baja, disminuyendo así el nivel de interferencia considerablemente, pero como desventaja, nos encontramos que el número medio de saltos entre origen-destino incrementa notablemente pudiendo provocar congestión en la red. Con ello llegamos a la conclusión que se ha de escoger cuidadosamente el nivel de potencia de transmisión para así obtener una buena eficiencia en el sistema.

Para que las simulaciones que a continuación detallaremos han de quedar claros los siguientes conceptos que a continuación se detallan:

- Dos nodos se escuchan, si la distancia entre ellos esta dentro del rango de transmisión predefinido.

- Se estudia el impacto del rango de transmisión en la conectividad, entendiendo la conectividad como la proporción de destinos que pueden alcanzar un nodo a través de simples o múltiples saltos.

- Se asume un modelo de red ideal donde un enlace puede ser establecido entre dos nodos cualquiera que esté dentro del rango de transmisión, y donde un camino puede siempre ser encontrado entre dos nodos conectados mediante una cadena de enlaces.

Una vez comentado esto, pasamos a comentar una simulación válida para todos los algoritmos, que se realizó en el artículo [3]. En el gráfico siguiente (**Fig.1.7**) se puede visualizar la conectividad media. En él se puede observar la relación entre el rango de transmisión y la conectividad media, para tener como resultado la garantía de que todos los nodos se puedan comunicar entre ellos teniendo cierto rango de transmisión. Como ejemplo podríamos ver, con un rango de transmisión alrededor de 30, N= 40, siendo N el número de nodos que tenemos en la red, que la conectividad es del 100%.

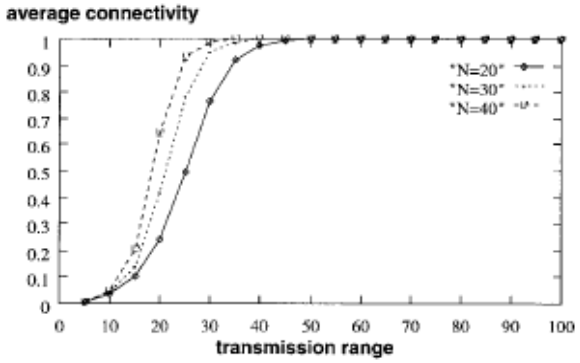

Fig. 1.7: Conectividad

1.4.1. Análisis comparativo de los algoritmos de clustering.

A continuación pasaremos a comentar las simulaciones realizadas para el HCC, aprovechando al mismo tiempo para realizar una comparación de este algoritmo con el algoritmo Lowest-ID (comentado también en el anterior apartado (1.2.1)), y el algoritmo de clustering distribuido presentado en [3].

En una red dinámica, los nodos pueden cambiar de localización, pueden desconectarse, o pueden asociarse. Un cambio topológico ocurre cuando un nodo se conecta o desconecta de todos o parte de sus vecinos, provocando una alteración en la estructura del cluster. Veamos el ejemplo de la siguiente figura (**Fig.1.8**)

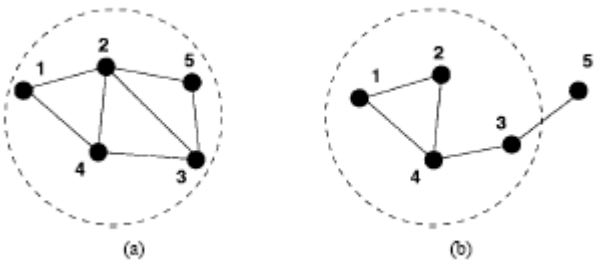

Fig.1.8: Reclustering

En la figura (**Fig.1.8(a)**) se observan que hay cinco nodos formando el cluster y que la distancia máxima de saltos no es más de dos. Como vemos en la figura (**Fig.1.8 (b)**) uno de los nodos se mueve cambiando como consecuencia, la topología anterior, teniendo ahora una distancia entre los nodos 1 y 5 ó nodos 2 y 5 igual a tres saltos, siendo mayor a dos que es el número máximo de saltos existentes en un cluster usando dichos algoritmos, teniendo que pasar como consecuencia, a reconfigurar el cluster.

En las simulaciones realizadas en el artículo [1] se observaron los cambios de conectividad, y las re-elecciones realizadas por los nodos, cuando éstos están en movimiento, algo que era bastante lógico, puesto que si no se movieran los nodos, no se podría observar la dinámica que tienen los algoritmos en cuestión.

Dos fueron las medidas monitorizadas durante el movimiento de los nodos, las cuales fueron:

- El número de nodos que cambian su estado a clusterhead.
- El numero de nodos que cambian de cluster.

En la simulación realizada se hicieron dos tipos de experimentos en función del tipo de movimiento que iban a tener los nodos, así pues, se realizaron pruebas para cuando el movimiento de los nodos era totalmente aleatorio, y para cuando la trayectoria del nodo estaba predefinida. Los resultados obtenidos fueron muy similares, llegando a la conclusión de que el tipo de movimiento ya sea aleatorio o predefinido, nos es totalmente indiferente, ya que siendo de una forma u otra, los cambios en nuestra arquitectura quedan reflejados independientemente del movimiento. Es por este motivo, por el cual solo reflejaremos en este documento la gráfica que se obtuvo con datos del movimiento aleatorio (**Fig.1.9**).

Los resultados obtenidos se muestran en la figura siguiente (**Fig. 1.9)**, usando para la realización de las pruebas 30 nodos.
Como se muestra en la figura, en el eje de ordenadas se muestra el número medio de cambios de cluster por unidad de tiempo. Y el eje de coordenadas nos muestra el rango de transmisión de los nodos, es decir, el abasto que tiene un nodo sin necesidad de repetidores.

Por otra parte, podemos ver que con HCC hay un mayor número de cambios, ya sean por nodos que migran a otro cluster, como en cambios del clusterhead, que con el algoritmo Lowest-ID. Con éste último se observa un número menor considerable de cambios que con el algoritmo HCC, proporcionando incluso una formación del cluster más estable. Esto es debido a que con HCC en función de los nodos asociados que tenemos podremos ser clusterhead (comentado en 1.2.1). Al estar los nodos en movimiento, este valor es muy variable, provocando un número de cambios elevado, llegando a provocar cierta inestabilidad en el sistema. En cambio esto no ocurre con el algoritmo Lowest-ID, ya que independientemente del número de nodos que haya en su alrededor, como éste se basa en ID predefinido por el administrador, no tenemos tantos cambios de clustehead ni de clusters.

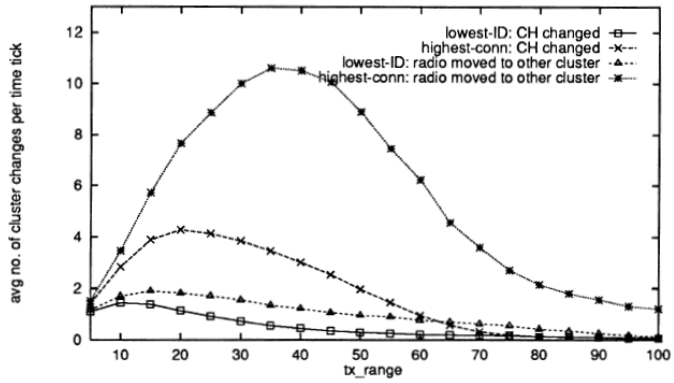

Fig. 1.9: Comparación clustering. Movimientos aleatorios

En las pruebas realizadas con el algoritmo de clustering distribuido presentado en [3], se medió la estabilidad de la arquitectura multicluster contando cuando nodos migran de un cluster a otro en intervalos de 100ms. En dicha simulación, cada 100ms todos los nodos se mueven en una dirección uniformemente distribuida dentro del intervalo (0,2π). En la figura (**Fig.1.10**) se muestra el número medio de nodos que cambian de cluster cada 100ms, viendo que los cambios de nodos son relativamente bajos dentro de un rango de transmisión de 40 a 50.

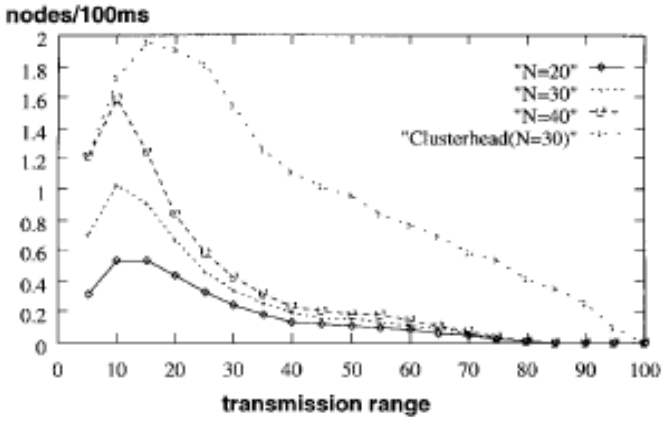

Fig.1.10: Estabilidad de la arquitectura multicluster.

A continuación se estudian las características de los clusters generadas por el algoritmo de clustering distribuido. En una arquitectura multicluster, hay nodos que hacen de gateways, los cuales retransmiten paquetes de un cluster a otro. El número de nodos pertenecientes a otros clusters (que también son repetidores) que puede escuchar un repetidor es variable, como ejemplo podemos ver en la **Fig.1.5** que el nodo 2 tiene dos clusters (el que pertenece y el del vecino) y el nodo 6 tiene cuatro (el que pertenece y tres más vecinos). En la figura siguiente (**Fig.1.11**) se puede ver el valor medio de clusters que ven los gateways en función del rango de transmisión. Mediante dicho grafico, se puede llegar a la conclusión que el número medio de clusters que ven los gateways está entre dos y tres.

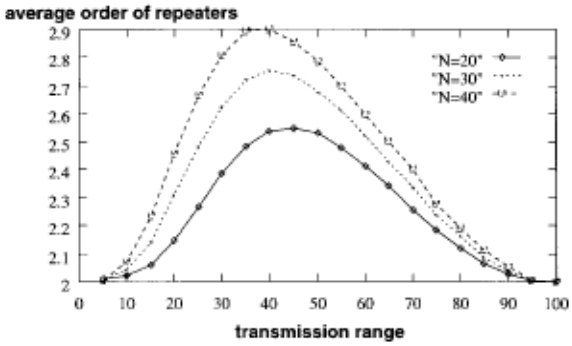

Fig.1.11: Media de órdenes a repetidores

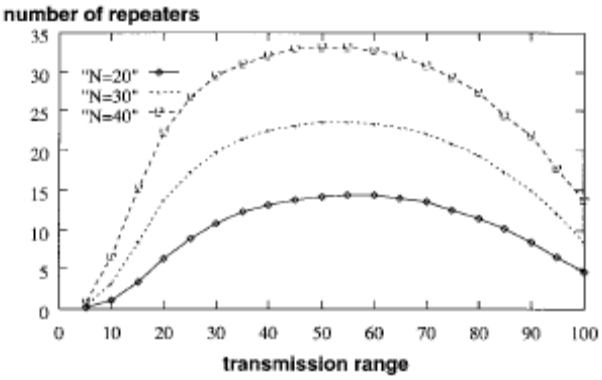

Fig.1.12: Número de repetidores

El número de gateways pueden afectar al número de caminos en la red. En la figura anterior (**Fig.1.12**) se puede observar que el 50% de los nodos son repetidores si el rango de transmisor está dentro del intervalo (30,80).

1.4.2. Análisis algoritmo MIX.

En el presente apartado pasaremos a explicar las simulaciones realizadas con el presente algoritmo de asignación de canales, el algoritmo MIX.
En el presente análisis, se han realizado dos tipos de simulaciones, una basada en una distribución de los nodos en una rejilla de 10x10 y otra, en una distribución de los nodos totalmente aleatoria en un plano de 200x200mts. A continuación pasamos a comentar la primera de las situaciones.

En la siguiente figura (**Fig.1.13**) podemos observar una rejilla de nodos de 10x10 con tres canales ortogonales. Para la presente simulación, el rango de transmisión fue fijado a todos los nodos por igual.

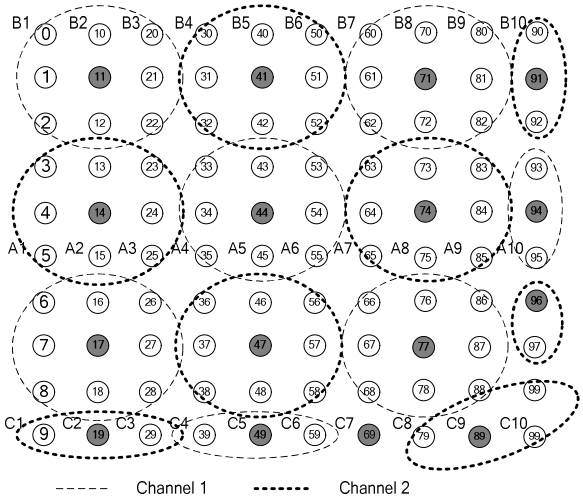

Fig.1.13: Rejilla de 10x10

Los nodos marcados de color más oscuro, se refieren a los CH, y los círculos indican rodean los límites del cluster. Como hemos mencionado anteriormente, para esta simulación se usan tres canales ortogonales, uno que será utilizado por la interfaz principal (pre-configurada por el administrador) que será el canal para las comunicaciones intra-cluster, y los otros dos canales serán usados para las comunicaciones Inter-cluster. La asignación de canales resultante, se observa en la figura (**Fig.1.13**) con los círculos punteados o líneas discontinuas.

En la siguiente figura (**Fig.1.14**), se realiza una comparación del rendimiento total a un salto entre la nueva arquitectura de clustering multi-canal con dos interfaz radio y la arquitectura tradicional de un canal, y una única interfaz radio. Claramente se observa una mejora del funcionamiento con clustering y múltiples canales ortogonales, teniendo una ganancia alrededor del 300%.

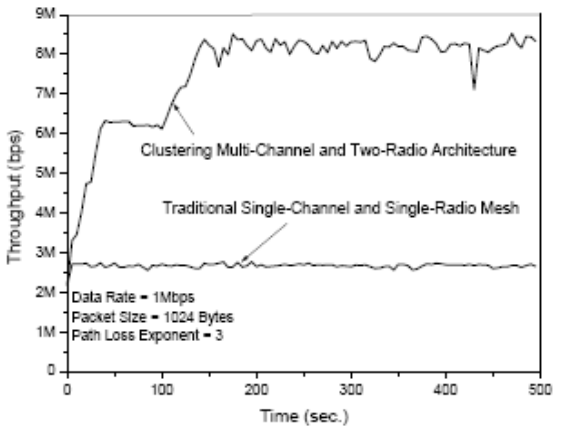

Fig.1.14: Comparación rendimiento a un salto.

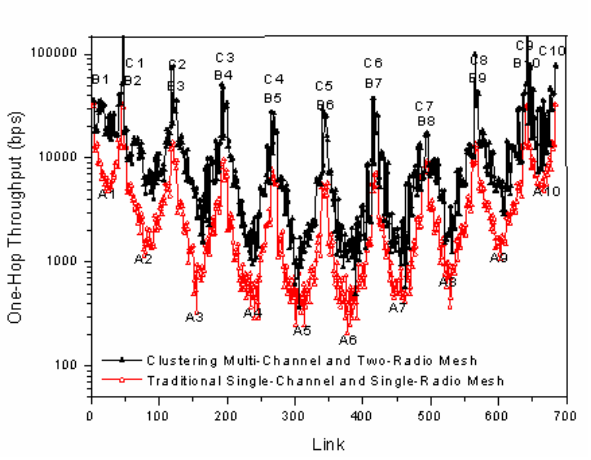

Fig.1.15: Distribución del rendimiento medio a un salto.

En el anterior gráfico (**Fig.1.15**), se muestra la distribución del rendimiento a un salto con respecto a los enlaces, donde los enlaces con A_i, B_i y C_i indicados en la figura (**Fig.1.13**), donde i = {1, 2, 3,…, 10}. Claramente se puede observar, que los enlaces A_i experimentan una fuerte interferencia de los enlaces B_i y C_i, provocando una oscilación en la distribución del rendimiento, ilustrando el problema de *imparcialidad dependiente de la posición*. Esto implica, por ejemplo, que las posiciones preferibles para las entradas en una red inalámbrica no pueden ser por el centro de la red.

Finalmente, tal y como hemos mencionado al principio de este punto, pasaremos a comentar una topología con una distribución totalmente aleatoria en un plano de 200x200mts (**Fig.1.16**). El rango de transmisión fijado para esta simulación es de 25mts.

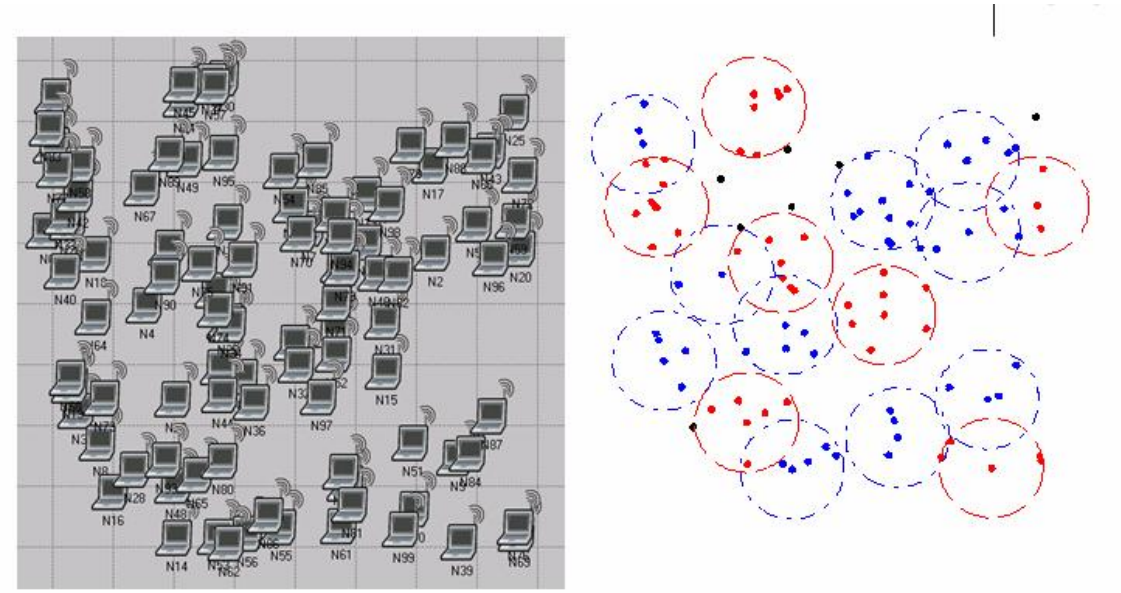

Fig.1.16: Topología aleatoria

Los colores rojo y azul indican los canales asignados a los nodos (canal 1 y 2) en sus respectivas interfaces radio secundaria; mientras que los nodos de color negro, son nodos que no pertenecen a ningún cluster.

En el siguiente gráfico (**Fig.1.17**) se compara el funcionamiento tanto con el rendimiento agregado como con la distribución de rendimiento. Claramente se ve que el rendimiento agregado de la arquitectura propuesta con 3 canales es casi tres veces más alto que el rendimiento tradicional.

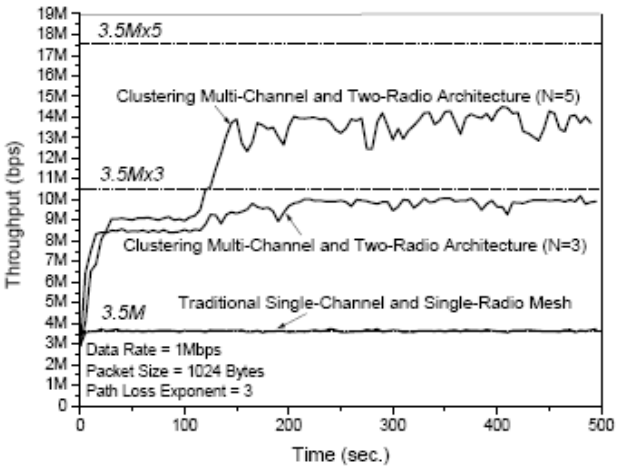

Fig.1.17: Rendimiento agregado.

Como se mencionó en la introducción, la reutilización espacial y canalizar la reutilización son la llave para una red de alto rendimiento. En la arquitectura propuesta de dos interfaces radio, la capacidad de reutilización espacial, sin embargo, se disminuirá como el número de canales ortogonales disponibles debido al tamaño de la red reducido por canal. En la figura (**Fig.1.17**) se

muestra el efecto de mayor número de canales. Claramente, el funcionamiento con 5 canales ortogonales es lejano debajo del grado óptimo, que sería 3,5Mbps x 5. Para obtener la misma capacidad de reutilización espacial aumentando el número de canales ortogonales tenemos que ampliar la arquitectura corriente de dos interfaces radio a una arquitectura multi-radio.

1.4.3. Análisis algoritmo DCA.

Las simulaciones realizadas [4] para el análisis del algoritmo DCA se han realizado comparándolo con otro algoritmo de asignación de canales, en concreto con el HP-CAM. Como métrica para el análisis de la comparación, se ha usado: energía consumida, mensajes transmitidos y colores requeridos.

En la siguiente figura (**Fig.1.18**) se comparan con el número toral de mensajes transmitidos para un número dado de nodos. Como se puede observar, el algoritmo DCA funciona considerablemente mejor que el HP-CAM. Para redes de altas densidades de nodos (1500 nodos), el algoritmo DCA transmite un 40% menos de mensajes.

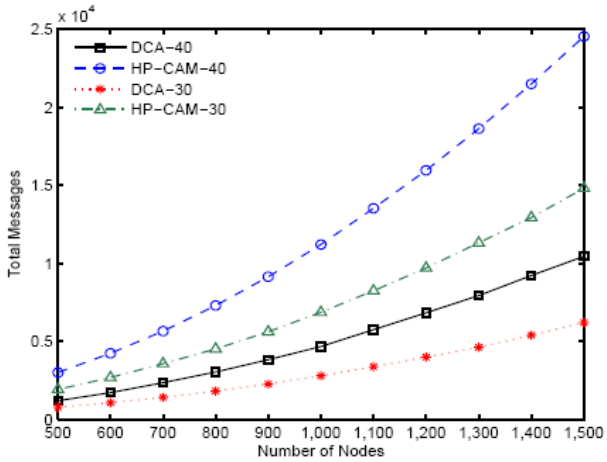

Fig.1.18: Número total de mensajes generados

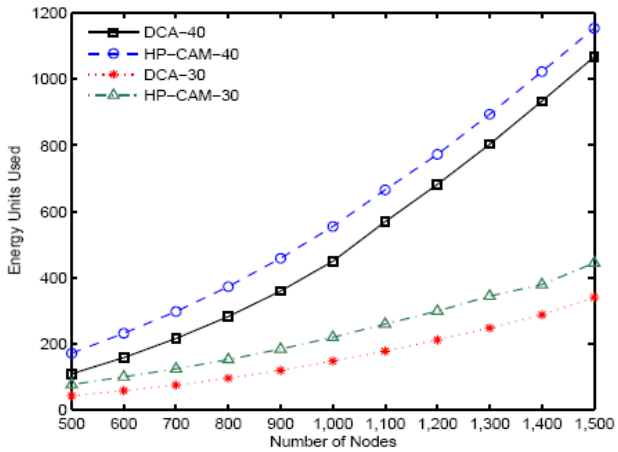

Fig.1.19: Consumo de energía medio.

En el anterior gráfico (**Fig.1.19**), se muestra una comparativa de la energía media consumida durante el proceso de coloring. Como se muestra, también hay una ligera mejora constante de ahorro de energía para toda la densidad.

A continuación (**Fig.1.20**), se analiza el número de mensajes de *CA (n(CA))* y *UP (n(UP))* generados por el algoritmo para una densidad de nodos dados. Se puede observar que *n(CA)* aumenta de manera más lenta que *n(UP)*. Esto ocurre porque cada cluster comprende cada vez un mayor número de nodos, causando como consecuencia un mayor número de mensajes de actualización para la asignación del canal.

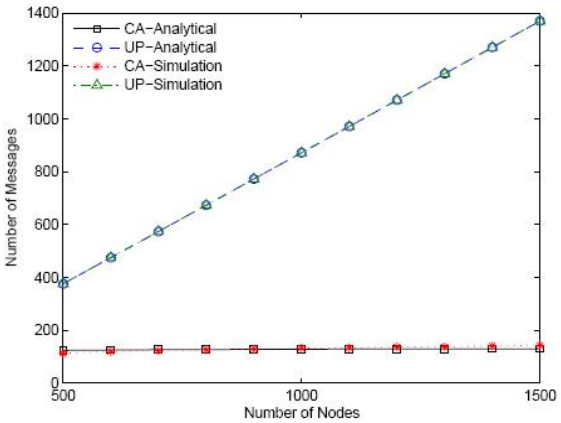

Fig.1.20: Comparación del número de mensajes de UP y CA

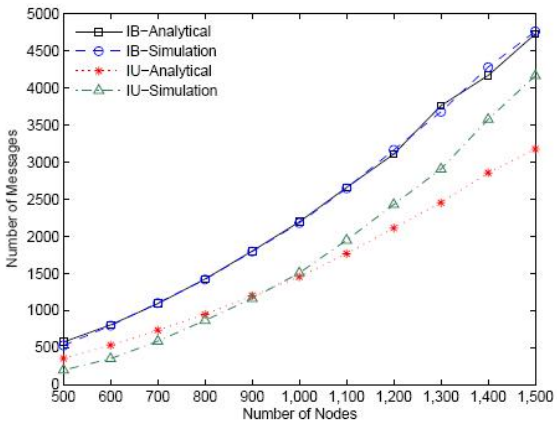

Fig.1.21: Comparación del número de mensajes IU e IB.

Finalmente, en la anterior figura (**Fig.1.21**) se muestra un gráfico del número de mensajes de INF Broadcast e INF Unicast, los cuales muestran un aumento esperado con la densidad del nodo y de acuerdo con la simulación y valores analíticos realizados.

1.4.4. Conclusiones

Una vez visto los análisis que se realizaron en los artículos [1][2][3][4] de los diferentes algoritmos anteriormente explicados, podemos sacar una serie de conclusiones, para posteriormente realizar una elección sobre el mecanismo en que se basará nuestra implementación.

En referencia a los algoritmos de clustering, tal y como hemos comentado en los análisis, el algoritmo Lowest-ID en una situación con nodos en movimiento, se comporta de manera mucho más eficiente que el algoritmo HCC. Cabe comentar, que no se puede realizar una comparación directa entre el Lowest-ID con el algoritmo de clustering distribuido, puesto que en el artículo [1] no especifica el intervalo de tiempo que se ha cogido para realizar las medidas del Lowest-ID. No obstante, el funcionamiento de ambos algoritmos es muy similar, como por ejemplo, ambos algoritmos se basan en la elección del clusterhead en el ID más bajo y no dependen de la cantidad de vecinos que tienen a su alrededor, cosa que pasaba con el HCC. Con lo cual, en una situación en la que los nodos se mueven aleatoriamente, se podría decir que el algoritmo de clustering distribuido, al igual que el Lowest-ID, es más eficiente que el HCC.

Referente a los algoritmos de asignación de canales, hemos podido observar que el funcionamiento de los algoritmos es totalmente diferente, no pudiéndolo comparar con ninguna de las graficas que figuraban en los artículos de los mismos [4][2], aun así a continuación sacaremos algunas conclusiones en referencia a estos dos algoritmos de asignación de canales.

Por lo que al DCA se refiere es un algoritmo pensado para dispositivos de una sola interfaz radio, y con un consumo relativamente bajo (**Fig.1.19**), de hecho

esta pensado su uso más bien para dispositivos de tecnología 802.15.4, conocida como Zigbee, para la creación de redes PAN. Con DCA se puede observar que los nodos se mandan una serie de mensajes para que la asignación de colores en los clusters sea correcta, pues bien, si nos encontramos con una red en la que los nodos se mueven aleatoriamente, los nodos han de estar continuamente mandando mensajes de control, perdiendo de esta forma eficiencia respecto los datos a transmitir de los diferentes usuarios conectados. A todo ello, si añadimos que en nuestra red puede estar formada por un número muy elevado de nodos, nos encontramos con un mecanismo poco escalable, ya que la eficiencia del mismo nos cae en picado.

Referente al algoritmo MIX, es el algoritmo más adecuado para un tamaño de red mayor (mayor número de enlaces, mayor distancia entre nodos, etc.) debido a que, como se comento en el apartado **1.3.1**, en este algoritmo el clusterhead para asignar el canal a utilizar en su cluster, mira en el medio que canal es el que menos interferencia recibe, para así asignárselo al cluster. Por otra parte, este algoritmo, esta pensado para trabajar con dos o más interfaces radio, aumentando así el rendimiento del enlace. Con este algoritmo hemos podido observar el aumento de rendimiento que existe con el hecho de utilizar una tecnología multi-canal con dos interfaces radio frente a la tecnología tradicional de un único canal y una única interfaz radio.

CAPÍTULO 2. DESARROLLO DEL ALGORITMO.

Tras estudiar las diferentes propuestas comentadas en el capítulo anterior, en el presente capítulo vamos a desarrollar un algoritmo en base a las ideas de los diferentes puntos estudiados, añadiendo además las mejoras oportunas que se han considerado necesarias para el desarrollo de un correcto funcionamiento del algoritmo.

En un primer apartado (**2.1**), se comenta cuál va a ser el funcionamiento final del algoritmo de clustering, en el que se comentan las mejoras añadidas frente al algoritmo de clustering distribuido que se ha escogido como referente.

A continuación, (apartado **2.2**) se comenta el algoritmo de asignación de canales que seguirán los diferentes nodos tras ya haber formado los diferentes clusters y haberse asignado los diferentes CH.

Para finalizar (apartado **2.3**), se ha creído conveniente el diseñar diferentes tramas para las diferentes situaciones en las que se encuentre el programa, para facilitar así la tarea del programador, y optimizar al máximo los tamaños de los paquetes, transmitiendo exclusivamente la información realmente necesaria.

2.1. Pseudo código Algoritmo de clustering.

Basándonos el algoritmo explicado en el apartado 1.2.2 se ha diseñado un algoritmo (**Fig.2.1**) para resolver ciertas limitaciones que encontramos en el mismo. Por ejemplo, se partía de una situación en la que se asume que todos los nodos ya conocen a sus respectivos nodos vecinos, aspecto que se ha corregido tal y como se puede ver en la **Fig.2.1**. Otro aspecto que se ha mejorado es que los datos que se almacenan en la tabla donde se guarda la información sobre los nodos vecinos están siempre actualizados durante el proceso de clustering, es decir, en caso de que se produzca un cambio, siempre se informarán a los nodos vecinos, modificando éstos sus respectivas tablas.

Una vez dicho esto, pasamos a explicar el pseudo código diseñado.

Todos los nodos al ser conectados por primera vez transmiten en broadcast su ID mas el ID_CH (que como se explicó, en un principio será su propio ID).

Para este algoritmo se entiende como vecino todos los nodos conectados a un salto del mismo. Todos los mensajes enviados en la ejecución de este algoritmo serán broadcast, para que todos los nodos sean informados de las posibles modificaciones del entorno.

Cada nodo, tendrá un vector de nodos dinámico en el que guardará información relativa a sus nodos vecinos, tanto su ID, como el CH de cada uno de ellos, para de esta forma realizar una elección de CH basándose en el valor de las IDs a partir del conocimiento del entorno que nos rodea. El vector se presenta la siguiente forma:

```
Typedef struct{
        Int id;
        Int id_ch;
}nodo;

nodo lista_vecinos[]
```

Gracias al vector dinámico no malgastamos memoria en los dispositivos que tienen características limitadas, aprovechando de manera eficiente de la memoria de que disponemos.

A diferencia del pseudo código del artículo [3], el cual partía desde el punto en que todos los nodos ya conocen a los nodos que les rodean antes de ejecutar el algoritmo de formación de clusters, en el presente proyecto se ha realizado un pseudo código capaz de tratar los datos al mismo tiempo que los va recibiendo, para de esta forma trabajar siempre con las listas actualizadas.

Si al finalizar el algoritmo, el CID es diferente al ID del nodo, el nodo pertenece a un cluster cuyo CH es el CID. Si por el contrario, el CID es igual al ID del nodo, ese nodo será CH del cluster, tomando todas las responsabilidades como tal.

```
        int x=0;
        int mi_ID;
        int mi_CID;
        char msg[];
        enviar = FALSE;
        typedef struct {
                int id;
                int id_ch
        }nodo;
        nodo lista_vecinos[];

enviar_mensaje (P, mi_ID, mi_CID)
set timer(5)

while true {
        escucha_mensaje (msg);

        x= guardar_nodos(lista_vecinos); //guardamos o actualizamos los mensajes de los nodos en
        lista_vecinos, y devuelve la posición de memoria donde lo guarda, o -1 en caso de no ser
        ninguna novedad.

        Si (x != -1) then reset timer

        si msg = P then
                        enviar = TRUE
        // Comparamos
```

```
        si (lista_vecinos[x].id == lista_vecinos [x].id_ch) { // Comprovamos si es CH
                si (mi_CID > lista_vecinos[x].id){
                        mi_CID = lista_vecinos[x].id        // Cambio mi CH
                        enviar = TRUE
                }
        }
        //Si mi CID deja de ser CH buscamos otro
        si (lista_vecinos[x].id == mi_CID && lista_vecinos[x].id =! Lista_vecinos[x].id_ch){
                mi_CID = mi_ID;
                buscar en lista_vecinos CH con menos id_ch
                enviar = TRUE
        }
        si enviar == TRUE then
                espera tiempo aleatorio
                enviar_mensaje (R, mi_ID, mi_CID)
}
si(mi_ID == mi_CID)
        {yo soy CH}
else
        {no soy CH}
falgoritmo
Alarm timer handler {
        exit algoritmo de clustering
}
```

Fig.2.1: Pseudo código algoritmo de clustering.

2.2. Algoritmo de asignación de canales.

Antes de comenzar a describir el diagrama de actividad presentado en la figura **Fig. 2.2**, pasaremos a comentar el comportamiento de un nodo desde el momento que se pone en funcionamiento, para así poder ver como a continuación nos adentramos al algoritmo dibujado.

En el momento que un nodo se pone en funcionamiento, a éste se le asigna un identificador (ID). A continuación, dicho nodo empezará transmitiendo su ID y el ID del clusterhead (ID_CH) al que pertenece (como en un primer estado, el nodo no conoce a ningún otro nodo, el ID_CH será su propio ID).

Una vez visto cómo se inicializan los nodos, estos empezarán a recibir ID's de los diferentes nodos vecinos, y al mismo tiempo ejecutarán el algoritmo de Clustering Distribuido explicado en el apartado **2.1** para la división de la red en clusters y la obtención de los CH de cada uno.

Una vez realizado el algoritmo de clustering, nos podemos encontrar con dos situaciones:

- El nodo ha sido seleccionado CH.
- El nodo no ha sido seleccionado CH.

Si un nodo no ha sido seleccionado como CH, éste difundirá más allá de las fronteras del cluster el ID_CH al que pertenece, para que así llegue dicha información a los CH's de los clusters colindantes.

Cuando un CH conoce los ID's de los CH's colindantes, el CH con un ID menor de entre sus vecinos asignará un canal para su cluster, y difundirá en broadcast su ID_CH y el canal escogido, para que los nodos del mismo cluster configuren su antena secundaria con el canal seleccionado por el CH. Estos a su vez deberán reenviar dicho mensaje a los clusters vecinos, para que así los CH's colindantes conozcan qué nodos CH han asignado canales, y qué canales han escogido.

En el momento que a otro CH le llega la información conforme un CH más prioritario que él ha asignado un canal en su cluster, éste comparará si no tiene CH's más prioritarios que él para así pasar a asignar un canal para su cluster. En el caso de tener que asignar un canal, éste asignará un canal que no interfiera con los canales escogidos por sus vecinos, es decir, que ningún nodo de su cluster sea interferido por nodos de clusters vecinos. Una vez el CH ha asignado un canal para su cluster, difundirá su ID_CH y el canal escogido, tal y como anteriormente hemos comentado.

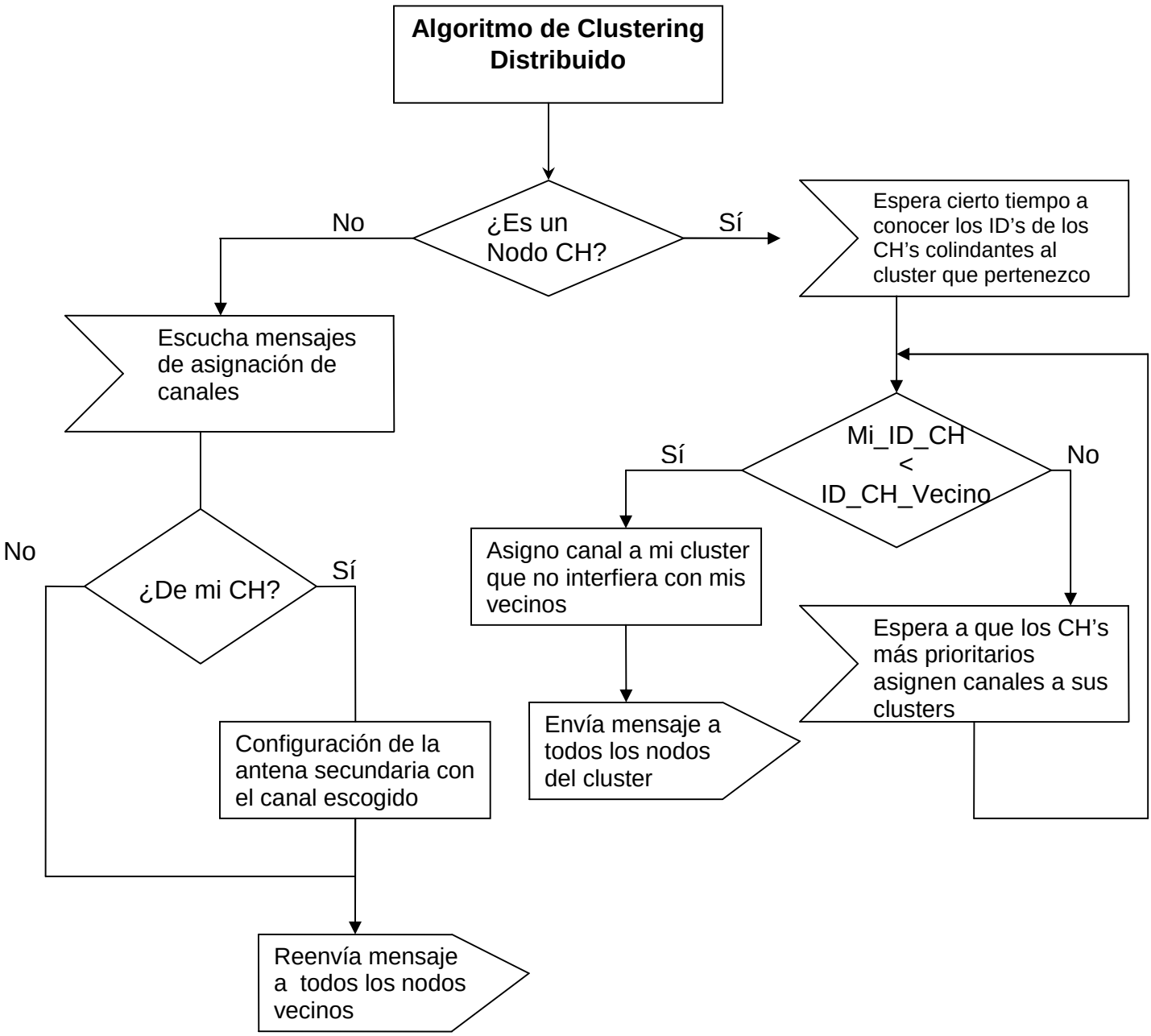

Fig.2.2: Diagrama de actividad

2.3. Formato de tramas usado para la implementación del algoritmo.

Para la implementación de dicho algoritmo se ha pensado en la creación de 3 tipos de tramas para el intercambio de mensajes entre los diferentes nodos a lo largo de la ejecución del algoritmo para que posteriormente la complejidad del programa sea menor y se obtenga una mayor eficiencia. Para la creación de dichas tramas se incorporarán una serie de campos en el campo de datos de la trama ethernet **Fig.2.3.**

@MAC_Dest	@MAC_Scr	Tipo Paquete	Datos	FCS
6Bytes	6Bytes	2Bytes	46-1500Bytes	4Bytes

Fig.2.3: Trama Ethernet

Los modelos de tramas definidos para cada punto de intercambio de mensajes serán las siguientes:

1. *Trama clustering*: Esta trama será transmitida a lo largo de la ejecución del algoritmo de clustering.

2. *Trama de información CH's vecinos*: Dicha trama se transmitirá hacia los CH de su propio cluster, para que éste conozca los ID's de los CH de los clusters vecinos.

3. *Trama canal asignado*: Tras conocer que el próximo cluster a colorear es al que pertenecemos, esta será la trama que envíe el CH a todos sus nodos vecinos para que configuren su antena secundaria. Ésta trama tambien será utilizada posteriormente para informar a los CH's vecinos del canal que se ha configurado por el CH que envia la trama, usándose en las siguientes situaciones:

    3.1. Tras colorear un cluster, los nodos pertenecientes al mismo informarán a sus clusters vecinos del canal que han escogido.
    3.2. Al recibir una trama de un nodo que ya ha coloreado informaremos al CH al que pertenecemos del canal que se ha seleccionado en el cluster vecino.

A continuación pasaremos a explicar todos los campos de las diferentes tramas definidas con más detalle.

2.3.1. Trama clustering.

La funcionalidad de los campos añadidos que se pueden observar en la figura **Fig.2.4** se explican a continuación:

Datos			
Tipo trama	**Tipo paquete**	**ID**	**ID_CH**
1Byte	1Byte	1Byte	1Byte

Fig.2.4: Trama clustering.

Tipo trama: Para la ejecución de los diferentes algoritmos que componen el protocolo de asignación de canales, se ha pensado en la creación de este campo, cuya finalidad como su propio nombre indica, será el comunicarnos qué tipo de trama es la que nos entrega, para así en el momento de recibirla, poder filtrar los paquetes de forma eficiente.

Tipo paquete: Este campo será el encargado de comunicar si esta trama se envía por primera vez (tipo “p”), o si es reenviada como respuesta a cambios detectados en la topología (tipo “r”).

ID: Como su propio nombre indica, en este campo viajará el ID del nodo que envía la trama.

ID_CH: El nodo que envía la trama, también ha de comunicar a qué CH pertenece. Para el caso en el que el mismo nodo es el CH, este campo llevará la misma información que el campo ID.

2.3.2. Trama de información CH’s vecinos.

Con este tipo de trama lo que se pretende es que el CH conozca los ID’s de los clusters colindantes que tiene a su alrededor, es por ello por lo que esta trama se transmitirá tras finalizar el Algoritmo de Clustering Distribuido por los nodos frontera (Gateways) hacia el CH de su propio cluster.

Datos		
Tipo trama	**ID_CH**	**ID_CH_vecino**
1Byte	1Byte	1Byte

Fig. 2.5: Trama de información de CH’s vecinos.

Como se puede observar en la **Fig.2.5**, los campos que precisaremos serán tres, el primero de los cuales ya ha sido comentado en el punto anterior.

ID_CH: En este campo indicaremos a quién va dirigida la trama, es decir, irá el ID del CH al que pertenece el nodo.

ID_CH_vecino: En este campo figurará el ID del CH de un cluster vecino, el cual se extraerá de la información recibida durante el proceso del algoritmo de clustering.

2.3.3. Trama canal asignado.

Tras conocer los ID's de los CH de los clusters vecinos, se pasa a asignar canales en los diferentes clusters. En el momento que a un CH debe asignar un canal, éste escogerá el más apropiado y difundirá la siguiente trama (**Fig2.6**) en la que indicará su ID y el canal escogido para que todos los nodos pertenecientes a dicho CH asignen el canal a su antena secundaria. El formato de la trama será como se describe a continuación (**Fig. 2.6**).

Datos		
Tipo trama	**ID**	**Canal**
1Byte	1Byte	1Byte

Fig.2.6: Trama canal asignado

ID: En este campo irá el ID del CH que informa a sus nodos vecinos qué canal es el elegido para dicho cluster, para que así todo aquel nodo cuyo CH sea igual a dicho ID, configure su antena secundaria.

Canal: En este campo vendrá reflejado el canal que se ha asignado para cubrir el cluster, el cual todos los nodos pertenecientes al mismo, deberán configurar sus antenas secundarias con el canal que se indica en dicho campo.

Llegados al punto en el que ya se ha seleccionado un canal para un determinado cluster, falta definir como se le va a pasar la información a los nodos CH de los clusters colindantes. Para la realización de éste paso, se ha pensado en la utilización de la misma trama que se ha comentado (**Fig.2.6**) puesto que nos es ideal para las siguientes situaciones que nos podemos encontrar.

- Situación 1: Un nodo ya ha asignado el canal que el CH ha escogido, y lo ha de reenviar a los nodos vecinos de los diferentes CH.

- Situación 2: Un nodo perteneciente a un cluster el cual esta a la espera de que sus clusters vecinos asignen canales, para que cuando llegue su turno pueda asignar el canal menos interferente en su cluster. En esta situación, los nodos frontera (gateway) estarán a la espera de que sus nodos vecinos no pertenecientes a su CH con

ID inferior al ID_CH nuestro, asignen un canal en el cluster al que pertenecen, y reciban la trama comentada en la situación anterior, para que ellos la reenvíen al CH que pertenecen.

CAPÍTULO 3. IMPLEMENTACIÓN Y EXPLICACIÓN DETALLADA DEL PROGRAMA.

La finalidad del presente apartado es explicar detalladamente el funcionamiento del algoritmo mediante un ejemplo, entrando en una explicación detallada de los puntos más importantes del algoritmo. Este apartado lo dividiremos en tres sub-apartados los cuales coinciden con la evolución que hemos ido haciendo a lo largo de la programación del mismo.

Antes de pasar a explicar el programa, comentar que los sockets utilizados para la implementacion del programa [5] han sido de nivel 2 conocidos como Raw-sockets o paquet sockets. Los conectores de paquetes (packet sockets) se usan para recibir o enviar paquetes directos (raw) en el nivel del driver de dispositivo (Nivel 2 de OSI). Permiten al usuario implementar módulos de protocolo en el espacio de usuario por encima de la capa física. La documentación de packet sockets se ha referenciado únicamente del manual de programador de Linux “Packet” [6].

En la siguiente figura (**Fig.3.1**) se observa una red con una configuración sencilla de nodos a partir de la cual iremos explicando el funcionamiento el programa.

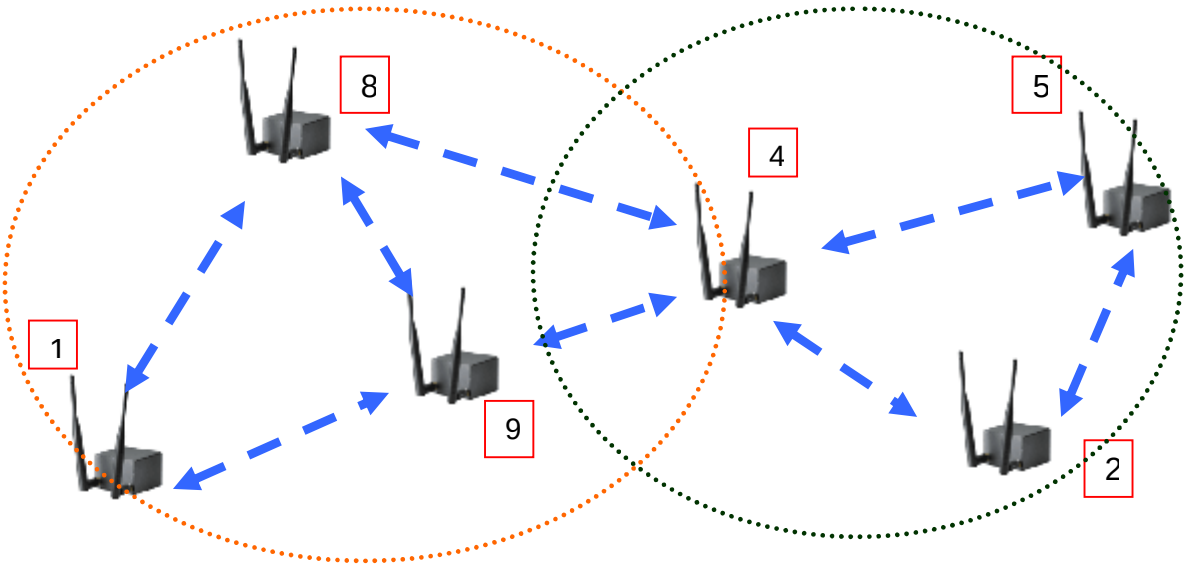

Fig.3.1: ID’s red mesh

3.1. Algoritmo de Clustering Distribuido

La finalidad del presente bloque, tal y como se explicó en el apartado 2.1., es dividir la red agrupando los nodos, formando así la creación de varios clusters para que posteriormente se puedan asignar canales de forma eficiente.

Antes de entrar en detalles del funcionamiento, cabe decir que en el momento en que un nodo se conecta, se le ha de dar un valor, este valor será el que posteriormente sea el ID de cada uno de los nodos. Supongamos pues, que en este caso tenemos una red con los siguientes valores[1] que se muestran la figura (**Fig. 3.1**).

A continuación pasamos a explicar la puesta en marcha del sistema.

Primeramente, tal y como hemos comentado al iniciar el nodo, nos pedirá un identificador, el cual será invariable hasta que el nodo se apague. Tras pasarle este indicador, el nodo que no sabe que es lo que tiene a su alrededor se da a conocer mandando un mensaje tipo *Trama Clustering* ya explicada en la **Fig.2.3**, donde *ID* es el identificador que le hemos pasado y el *ID_CH* será como se puede ver en el pseudo código el mismo valor que el ID, puesto que al iniciarse un nodo, éste se inicializa tomándose él por CH.

Tras haber mandado el mensaje dándonos a conocer en la red, entramos en un bucle (**Fig.3.2**) en el que estaremos continuamente escuchando el medio preparados para recibir cualquier información de cualquier nodo que nos pueda ser válida para la formación del cluster. Cada nodo dispone de una tabla en la que guardará la información que sus vecinos envían en este proceso. En esta tabla únicamente se guardan los valores *ID* e *ID_CH*, así sabemos en todo momento los nodos que nos rodean. Éste bucle finaliza tras pasar 10 segundos sin cambios en el entorno.

```
signal(SIGALRM,catch_alarm);
alarm(10);

        while(1){
                if(recibe_msg(mens)==0 ){       // esperamos a la recepcion de un mensaje
                                                // si es = 0 significara que es de nuestro
                                                //    protocolo
                        …
                        …
                }
        }
```

Fig.3.2: Bucle infinito

[1] Valores tomados estratégicamente para la realización de todos los aspectos.

Cada vez que agregamos un nuevo nodo en la tabla o recibimos una modificación de algún nodo vecino (en este paso, cuando hablamos de modificación de algún nodo, nos referimos a que ha cambiado su *id_ch*), ejecutamos a continuación el algoritmo de clustering explicado en 2.1., ya que de esta forma, si nos llega información de alguno de nuestros vecinos, indicándonos que es CH, y que tiene un ID más bajo que nuestro *ID_CH*, cambiaremos nuestro *ID_CH*, e informaremos mediante un mensaje broadcast a todos los nodos de nuestro alrededor (Véase el pseudo-código **Fig.2.1**), teniendo que modificar de esta forma la tabla, y en según que casos tener recorrer toda la tabla para asociarse a otro CH, o asignarse el cargo de CH.

Veámoslo con el siguiente ejemplo:

Supongamos que partimos de cero, y el primer nodo en conectarse es el nodo 9, éste enviará su *ID* y su *ID_CH*. Puesto que no hay ningún otro nodo en funcionamiento, éste nodo se quedará en espera de recibir respuesta de cualquier otro nodo al que le haya llegado el mensaje. Antes de que pasen los 10 segundos que un nodo está en espera de recibir mensajes sin ninguna modificación, conectaremos el nodo 8. Éste enviará una trama informando quién es. El nodo 9 recibe dicha trama y guardará los campos *ID* e *ID_CH* en una tabla donde se almacenan todos los nodos vecinos que tenemos a nuestro alrededor. Tras guardar los valores, pasaremos a la ejecución del algoritmo de clustering comprobando si el nodo que hemos almacenado es un CH, y si es así comprobar que el *ID* de dicho CH es menor que mi *ID_CH*, con lo cual, el nodo 9 que hasta el momento se tomaba como si fuera CH, pasará a tener un *ID_CH* igual a 8 y enviará un mensaje broadcast para que todos sus vecinos tengan constancia del cambio producido. En este momento, al nodo 8 que estaba en espera de la recepción de algún mensaje, le llega la información del nodo 9, y éste tal y como anteriormente ha hecho el nodo 9, pasará a guardar la información que dicho nodo le ha pasado. A continuación dicho nodo mirará si el nodo que le ha enviado la información es CH o no, en este caso no es CH puesto que los campos *ID* e *ID_CH* no son los mismos y pasaremos a la espera de recibir mensajes.

A continuación pasaremos a conectar el nodo 4. Como hemos mencionado anteriormente, este enviará una trama indicando que es el nodo 4 y él es CH. Esta trama le llega al nodo 8 y al nodo 9. El nodo 9 guardará el mensaje en su tabla, y mirará si el destinatario del mensaje es CH, como si que lo es, comparará si el ID de dicho nodo es inferior que el *ID_CH* que tenemos nosotros, y entonces cambiaremos el *ID_CH* ya que 4 es menor que 8. Al modificar el *ID_CH* del nodo 9 enviaremos un mensaje broadcast para que todos los nodos vecinos tengan constancia de dicha modificación.
El nodo 8 hará lo mismo que el nodo 9, y además dejará de ser CH puesto que a partir de ahora será el nodo 4 quien tome las riendas. Al nodo 8 le llegará la modificación que ha hecho el nodo 9 referente al *ID_CH* del mismo y la modificará en su tabla, al igual que el nodo 9 hará la respectiva modificación referente al nodo 8. Al nodo 4 le llegara la información de ambos nodos, y lo único que hará será almacenarla en su tabla, puesto que ninguno de los dos es

CH actualmente. Llegados a este punto las tablas de dichos nodos (**Fig.3.3**) serán:

Nodo 9	
ID	**ID_CH**
8	4
4	4

Nodo 8	
ID	**ID_CH**
9	4
4	4

Nodo 4	
ID	**ID_CH**
8	4
9	4

Fig.3.3: Tablas de guardar_nodos

Este proceso se va repitiendo continuamente hasta que pasen 10 segundos sin que los nodos no reciban ninguna modificación o ninguna agregación nueva de algún nodo. Pasado dicho tiempo el thread responsable de establecer el cluster es finalizado (**Fig.3.4**), saliendo así del proceso de clustering.

```
void catch_alarm(int sig_num){
        alarm(0);
        printf("****Salimos del Clustering****\n");
        pthread_exit(NULL);
}
```

Fig.3.4: Función de salida del clustering

A continuación (**Fig.3.5**) podemos observar el intercambio de mensajes total que se producirán entre los diferentes nodos. Para la realización de dicho ejemplo hemos realizado la conexión de los nodos de mayor a menor.

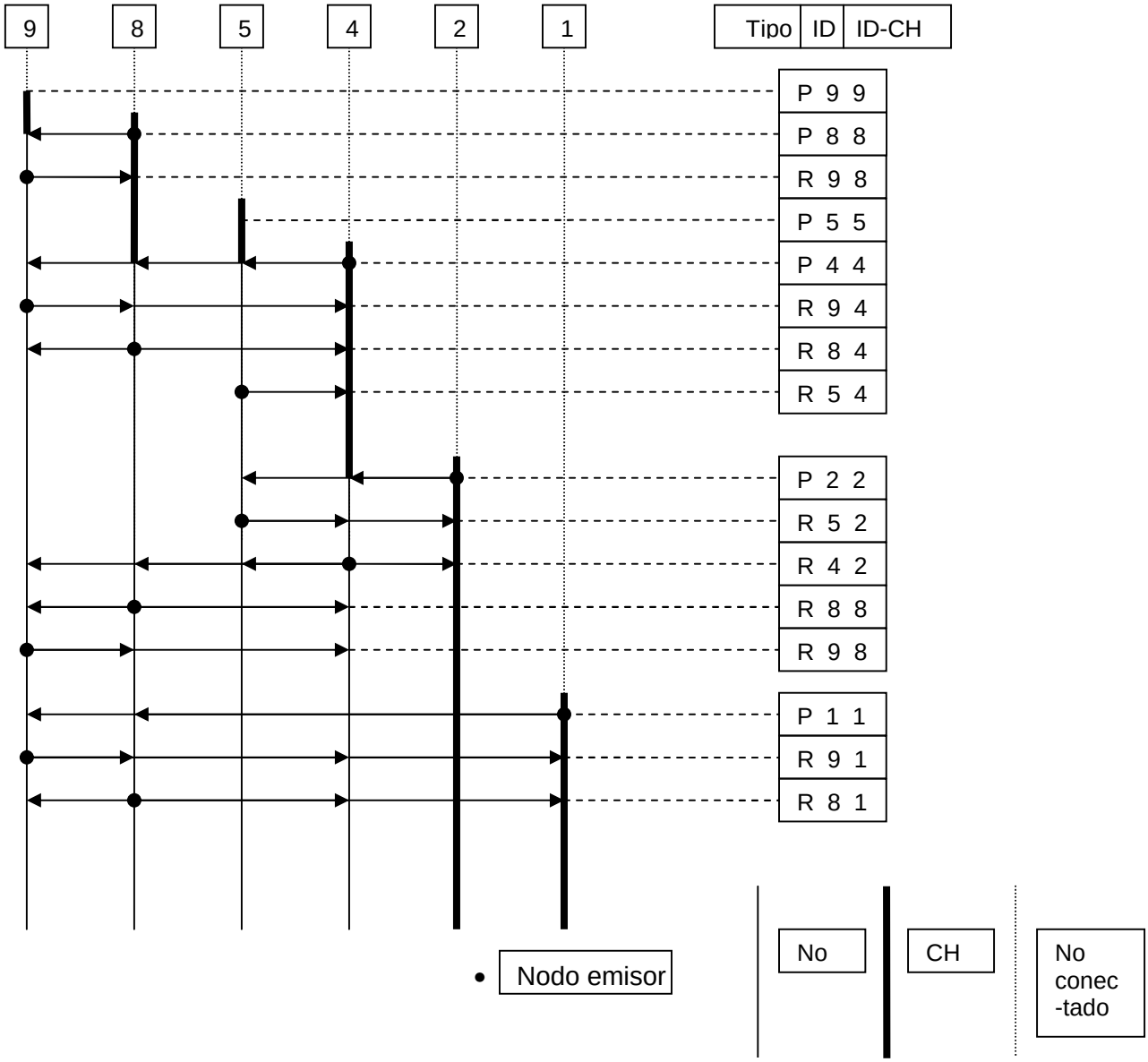

Fig. 3.5.: Diagrama completo de secuencia

3.2. Fase de conocimiento de los CH de los clusters colindantes.

Una vez visto como se forman los clusters, el siguiente paso con el que nos encontramos es que los CH de cada uno de los clusters formados conozcan los ID's de los CH de su alrededor.

La forma que se ha pensado para llevar a cabo esta fase ha sido la siguiente:

- El nodo que sea CH, mirará su tabla lista_vecinos y mirará si alguno de los nodos que tiene almacenado, tiene un ID_CH diferente al su ID. Almacenando, en caso afirmativo, dicho valor en una estructura tipo **Fig.3.6.** A continuación quedará a la espera durante 15 segundos de la llegada de mensajes de información de los nodos que pertenecen a su cluster indicándole, si lo hay, los diferentes CH que ellos tienen almacenados en sus tablas. Los 15 segundos que se han establecido para la espera de recepción de mensajes es por si algún nodo acaba de realizar el algoritmo de clustering más tarde que el CH (ya sea porque vea a más nodos que el CH de dicho cluster y se realicen más incorporaciones a última hora) y así evitamos perder la información que este posteriormente hará circular.

```
typedef struct Tch_vecino{
        unsigned char id;
        unsigned char canal;
        unsigned char asignado;
}Tch_vecino;
```

Fig.3.6: Estructura Tch_vecino.

- Los nodos que no sean CH, como hemos comentado anteriormente, únicamente mirarán la tabla lista_vecinos que han rellenado en el proceso de clustering, y reenviarán al CH al que pertenecen todos aquellos ID_CH's que no sean igual al que ellos pertenecen (**Fig.3.7**).

```
Si (Mi_CID != lista_vecinos[x].id_ch){
        //Enviamos a nuestro CH la trama: Tipo_msg, Mi_CID, ID_CH_Vecino
        enviar_mensaje(datos);
}
```

Fig.3.7: Reenvio mensaje

3.3. Asignación de canales.

Tras haber formado los clusters, y conocer los ID's de los CH's que hay en los clusters vecinos, únicamente nos falta asignar los canales de una forma eficiente evitando interferencias con los clusters vecinos.

Empezamos comentando como un CH asigna un canal para su cluster, y más adelante acabamos comentando el comportamiento del resto de nodos.

Llegados a este punto, el nodo que sea CH, primeramente ordenará su tabla *Tch_vecinos* de menor a mayor en función del ID (en el caso que tenga vecinos más de 1 CH vecino en dicha tabla).

Seguidamente contará cuantos CHs tienen ID's inferiores al ID del mismo. Si no hay ningún nodo que tenga un ID inferior al ID del nodo en cuestión, dicho nodo será el primero en asignar canal (**Fig.3.9**).

```
…
for (x=0; x<cont; x++){
        if (Tch_vecinos[x].id < cid){
                // Incrementamos contador contando de esta forma
                // cuantos vecinos tenemos mayores que nuestro ID_CH
                aux++;
        }
}
if (aux == 0){
        // asigno canal
…
```

Fig.3.9: Función búsqueda ID's inferiores

Al ser dicho nodo el primero en asignar un canal para su cluster, tiene total libertad para escoger el canal por el que desea transmitir, con lo que siempre por defecto se escogerá el canal 1 si no se detectan interferencias de otros sistemas.

Si se encuentran ID's inferiores al nodo en cuestión, entraremos en un bucle del cual sólo saldremos una vez hayamos asignado el canal (**Fig.3.10**). Dentro de dicho bucle, el nodo espera a recibir mensajes de información de los CH que asignan canales. Tras recibir un mensaje se guardará la información correspondiente en la tabla *Tch_vecinos* y marcará el campo *asignado* a 1 (este campo se diseñó para poder hacer filtros). A continuación volveremos ha realizar una búsqueda de ID's inferiores al ID del nodo como la que se ha realizado anteriormente, pero filtrando aquellos nodos que ya han asignado canal, pudiendo llegar de esta forma al momento en el que el nodo pueda asignar un canal para dicho cluster.

```
Mientras (no sea mi turno){

        if (recibe_msg(mens) ==0){      //esperamos la recepcion de mensajes de los Clusters
                                                con id_ch menor se asignen un canal

                // Guardamos el canal que han asignado en la tabla
                        Tch_vecinos y  marcamos el campo asignado a 1

                //Comprovamos si no hay ningun otro CH mas
                        prioritario para asignar canal

                if (es mi turno){
                        //asigno canal
                …
```

Fig.3.10: Función de espera asignar canal

Una vez al nodo le llega el "turno" de asignar canal, éste a diferencia de antes, no podrá escoger cualquier canal aleatoriamente, sino que tendrá que asignarse un canal que no esté siendo usado por ninguno de sus clusters vecinos. Para ello cuando a un nodo le toca escoger un canal, se asignará por defecto el canal 1, e irá comparando dicho canal con los canales que ya han asignado los CH vecinos. En el caso de que otro CH ya haya asignado dicho canal para su cluster, el nodo incrementará el valor del canal y volverá ha hacer dicha comparación, hasta encontrar un canal libre.

Los nodos que no son CH, se mantendrán a la espera de que el CH al que pertenecen les envíe una trama de asignación de canal (**Fig.3.11**) para así ellos poder configurar su antena secundaria. Hasta el momento en que estos nodos no configuren su antena secundaria, todos los mensajes de asignación de canal que reciban de nodos vecinos que indiquen el canal que han configurado un determinado cluster, será reenviado para que el CH al que pertenece guarde dicha información en la tabla *Tch_vecinos*, de modo que posteriormente el CH pueda ejecutar correctamente la función de asignar canal.

```
si (no soy CH){

        mientras ( MI_CH no asigna canal ){
                si (recibe_msg(mens) ){
                        si (origen_msg == MI_CID){    //si el mensaje ha sido generado
                                                              por nuestro CH,configuramos
                                                              la antena secundaria
                                configura_antena_2;
                        }
                        sino{    //mientras estamos a la espera de que nuestro CH nos envie el
                                        canal para el cluster, vamos reenviando las tramas que
                                        vallamos cogiendo
                                reenviar_mensaje();
                        }…
```

Fig.3.11: Función nodo no CH esperando configurar canal.

Veamos a continuación un ejemplo.

Tomando como ejemplo la **Fig. 3.1,** y situándonos en el punto en que ya se ha realizado el algoritmo completo de clustering, y cada CH ya conoce quienes son sus CH's vecinos, pasaremos a comentar como asignan los canales el nodo 1 y el nodo 2.

Empecemos comentando el nodo 1. Tal y como hemos mencionado anteriormente, lo primero que se hará es ordenar la tabla en el caso que el número de CH vecinos sea mayor de 1, pero como solo tenemos 1 CH vecino para dicho ejemplo, no haremos ninguna ordenación. Seguidamente, el nodo 1 contará cuantos CHs vecinos tiene en su tabla con un ID inferior al suyo. Para este primer caso, dicho nodo va ser el nodo que tiene un ID inferior, con lo que pasará a asignar un canal directamente. El canal que asignará será el canal 1 puesto que es el que se ha definido por defecto, y enviará un mensaje a todos sus nodos vecinos informando qué canal se ha escogido, para que así los nodos pertenecientes a dicho CH configuren su antena secundaria y reenvíen el mensaje a nodos vecinos pertenecientes a otros clusters.

El nodo 2 ordenará la tabla, y a continuación buscará si hay algún CH que tenga un ID inferior al suyo. A diferencia de antes, el nodo 2 observa que el nodo 1 es inferior. El nodo 2 se mantendrá a la espera de la recepción de algún mensaje de *información de canal asignado*. En el momento que recibe dicho mensaje que le será proporcionado por el nodo 4, guardará en la ficha del nodo (*Tch_vecinos*) el canal que ha asignado y marcará el campo *asignado* a 1 para indicar que ya ha asignado canal. Seguidamente volverá a recorrer la lista *Tch_vecinos* filtrando todos aquellos nodos que ya han asignado canal, y veremos que el siguiente en asignar un canal es el propio nodo 2. Por defecto también se asignará el canal 1, entonces mirará en la tabla *Tch_vecinos* los nodos que ya han escogido canal y comparará el canal escogido con los que ya están seleccionados, para así evitar interferencias. Para esta situación, el nodo 2, verá que el canal que había escogido, ya está siendo usado en otro cluster, y cambiará el canal a usar por el canal 6. Volverá a recorrer la lista viendo en esta ocasión que ningún vecino ha asignado el canal 6 para su cluster, y se lo asignará definitivamente, enviando a continuación un mensaje broadcast a todos los nodos vecinos para que, tal y como hemos mencionado anteriormente, todos los nodos de su cluster configuren su antena secundaria y reenvíen dicha información a nodos vecinos no pertenecientes al cluster.

Capitulo 4: Pruebas de rendimiento

En el siguiente capítulo se ha hecho una comparativa real del montaje de diferentes topologías, para poder ver las diferencias obtenidas usando un único canal de transmisión o usando de manera eficiente los canales de los que se disponen con la tecnología 802.11. La comparativa se basa en la realización de medidas de retardo y ancho de banda.

Para la realización de todas las pruebas se ha configurado en cada uno de los nodos las rutas de manera estática, siendo cada enlace una red diferente, de forma que forzamos que a la hora de mandar un paquete se esté obligado a pasar por cada uno de los cubos. De ésta forma podremos sacar conclusiones en función de los resultados obtenidos.

Para caracterizar el retardo extremo – extremo de los escenarios se va a utilizar la utilidad ping. Para ello veremos que opciones nos ofrece:

Ping 192.168.2.2 –c 20 –s 56

Con –c se indica el numero de paquetes que se van a enviar.
Con –s se indica el tamaño de los paquetes que se van a enviar.

Para la caracterización del ancho de banda utilizaremos la herramienta IPERF y retransmitiremos paquetes TCP por su capacidad de llegar a transmitir a la máxima tasa posible de manera automática. El efecto producido por TCP ante las pérdidas de tramas típicas de un entorno radio, permitirá observar más claramente la diferencia entre el rendimiento con un solo canal/interfaz y con el uso de clustering.

Para ello las opciones utilizadas en el lado del cliente han sido:

Iperf –c 192.168.2.2 –M 1000 –t 60

Siendo –M el tamaño del segmento en Bytes, y –t el tiempo en segundos de la medida.

En la siguiente figura (**Fig. 4.1**) se muestra un ejemplo de los resultados que nos dá usando esta herramienta:

```
root@zalux2:/home/dani # iperf -c 192.168.2.2 -M 1000 -t 60
WARNING: attempt to set TCP maximum segment size to 1000, but got 536
------------------------------------------------------------
Client connecting to 192.168.2.2, TCP port 5001
TCP window size: 16.0 KByte (default)
------------------------------------------------------------
[  3] local 192.168.4.1 port 37976 connected with 192.168.2.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-60.1 sec  13,01 MBytes   1,73 Mbits/sec
```

Fig.4.1: Cliente

Para la realización de las medidas se ha usado un tiempo elevado para poder despreciar el efecto del slow-start (característica que nos ofrece TCP cuando empieza a transmitir) para que así las medidas sean lo más exactas posibles.

4.1. Prueba 1: Conexión de nodos en línea.

La primera prueba que se ha realizado ha sido implementando el siguiente escenario (**Fig.4.2**), donde se sigue una topología en línea.

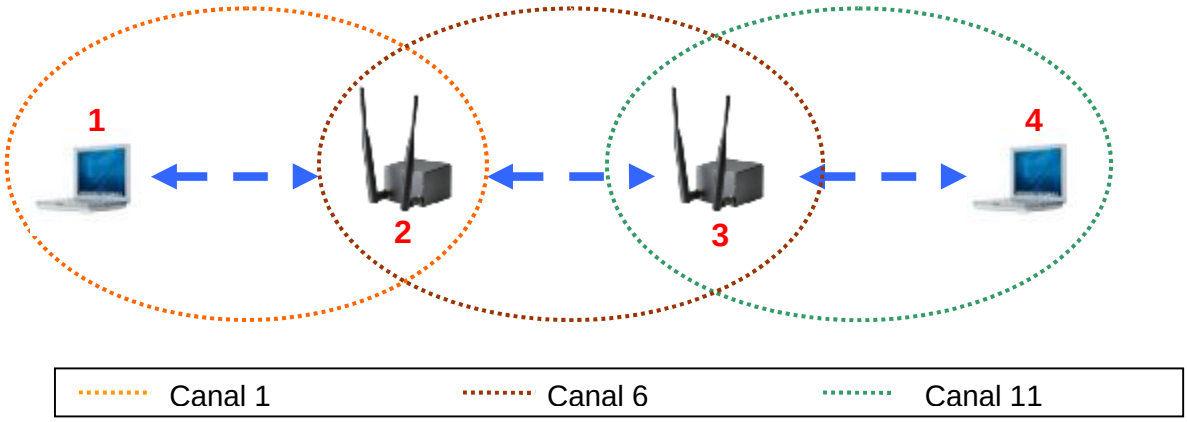

Fig.4.2: Escenario 1

4.1.1. Pruebas con canales diferentes

La primera prueba con el presente escenario se ha realizado asignando diferentes canales para cada uno de los enlaces, tal y como se indican en la figura (Fig.4.1), de forma que no se produzcan interferencias entre ellos tras agrupar los nodos en dos clusters. El nodo 1, al ser CH selecciona el primer canal libre, el canal 1, mientras que el nodo 3 selecciona el 11 al estar siendo usado el 6 para las comunicaciones Inter-cluster. Cabe recordar que cada uno de los enlaces pertenece a redes diferentes, evitando de esta forma una visión directa extremo a extremo de dicho escenario.

4.1.1.1. Caracterización del retardo:

Tras la elaboración de múltiples pruebas con un paquete de tamaño de **56** Bytes de datos los diferentes retardos medios obtenidos fueron los siguientes (**Fig.4.3**):

Pr1	Pr2	Pr3	Pr4	Pr5	Media
7.167ms	7.173ms	7.019ms	7.461ms	7.354ms	**7.235ms**

Fig 4.3.: Resultados retardo

Tras varios intentos se observó que las diferencias eran mínimas dando como validos los resultados obtenidos.

Cabe comentar que los resultados obtenidos son RTT (de ida y vuelta), es decir, si quisiéramos saber exactamente el tiempo que tardamos en llegar de un extremo a otro, deberíamos de dividir dichos resultados entre dos.

4.1.1.2. Caracterización del ancho de banda:

Al igual que se ha realizado con el retraso, para la caracterización se han repetido las pruebas varias ocasiones, tomando como validos los resultados obtenidos.

Los anchos de banda obtenidos fueron los siguientes (**Fig.4.4**):

Pr1	Pr2	Pr3	Pr4	Media
1.72 Mbps	1.80 Mbps	1.73 Mbps	1.72 Mbps	1.743 Mbps

Fig.4.4: Resultado ancho de banda

Los anchos de banda obtenidos en esta prueba fueron bajos frente a los 4-5 Mbps que debería ofrecer la tecnología 802.11b en una transmisión TCP. Esto puede ser causa de que los cubos dispongan de una velocidad de conmutación interna bastante baja.

4.1.2. Pruebas con canales iguales.

Manteniendo el mismo escenario usado para la prueba anterior, se ha modificado los diferentes canales configurados en cada uno de los enlaces configurando todas las interfaces asignando un mismo canal común (Canal 1). El objetivo de esta prueba es de demostrar que realmente las interferencias que van a provocar entre un enlace y otro, provocan una caída en la calidad del canal importante, bajando las velocidades de transmisión del mismo.

4.1.2.1. Caracterización del retardo:

Primeramente, al igual que se ha realizado en el apartado anterior se ha medido el retardo obtenido de extremo a extremo de la red. En esta ocasión los retardos obtenidos han tenido una media de 8.31ms, superior a los 7.235ms

obtenidos asignando canales diferentes. Tal y como se ha comentado antes, este tiempo de diferencia es debido por causa de la interferencia recibida en los nodos.

4.1.2.1. Caracterización del ancho de banda:

A continuación se pasa a realizar las pruebas de rendimiento del escenario con canales iguales. Para esta ocasión se han tomado tres medidas (**Fig.4.5**) dándoles el visto bueno por su similitud, y puesto que es el orden que se esperaba.

Pr1	Pr2	Pr3	Media
900 Kbps	901 Kbps	798 Kbps	866 Kbps

Fig. 4.5: Resultados ancho de banda

Como se puede observar, son del orden de 1 Mbps inferiores a los obtenidos asignando canales diferentes en cada uno de los enlaces.

Una vez más, se puede afirmar la importancia que requiere la realización de una buena distribución de canales, intentando evitar las interferencias para poder alcanzar así el máximo rendimiento que los dispositivos proporcionan.

4.2. Prueba 2: Caracterización de los nodos en cuadrado.

La siguiente implementación se trata de la figura siguiente (**Fig.4.5**):

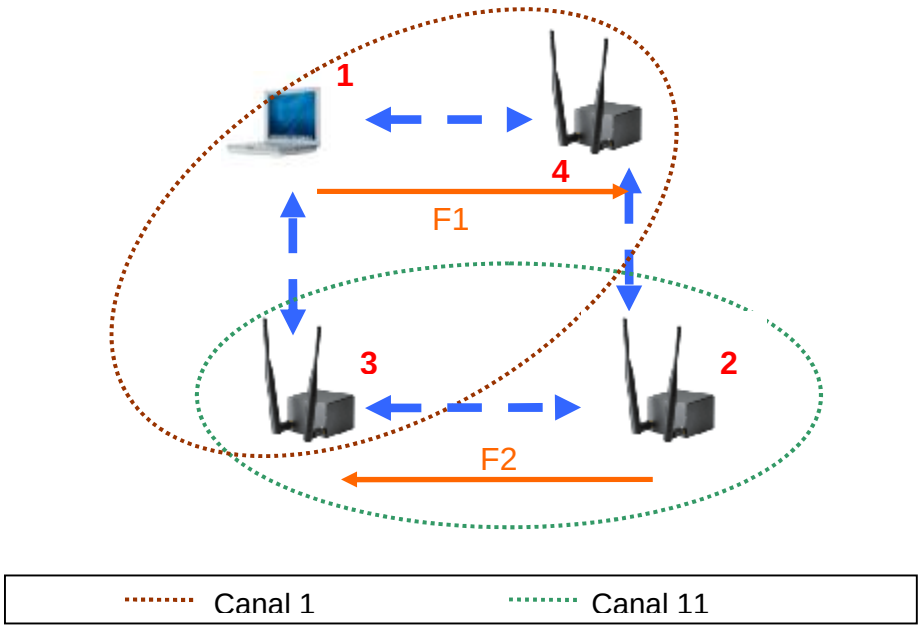

Fig.4.5: Escenario 2

4.2.1. Pruebas con canales diferentes

En la siguiente implementación, para la realización de las medidas se emitirán dos flujos simultáneos entre los nodos 1-4 (F1) y 2-3 (F2). Como es de esperar, ambos flujos serán totalmente independientes, es decir, no se interferirán entre ellos puesto que trabajan en bandas de frecuencia diferentes, teniendo que llegar a una eficiencia de la canal óptima.

4.2.1.1. Caracterización del retardo:

Nuevamente se hará uso de la misma herramienta comentada al inicio del capítulo para la caracterización del retardo, el ping.

Al igual que antes, el tamaño usado para el ping ha sido de 56Bytes, y en esta ocasión para todas las pruebas realizadas el valor medio obtenido ha sido siempre el mismo. Entre los nodos 1-4 el retardo ha sido 2.979ms (**Fig.4.6**) y entre los nodos 2-3 el retardo ha sido de 2.977ms.

```
--- 192.168.2.4 ping statistics ---
29 packets transmitted, 29 received, 0% packet loss, time 28024ms
rtt min/avg/max/mdev = 2.324/2.698/2.979/0.201 ms
```

Fig.4.6: Retardo entre nodos 1 y 4

4.2.1.2. Caracterización del ancho de banda.

En el siguiente apartado calcularemos el throughput máximo de dicho escenario (**Fig.4.5**). Para ello sumaremos los datos que se obtengan en el flujo producido entre los nodos 1-4 y se sumaran a los datos obtenidos en el flujo producido entre los nodos 2-3.

Para la realización de dicha prueba, se ha utilizado el IPERF con un intervalo de tiempo de 60 segundos. También tras realizar la prueba numerosas veces los resultados obtenidos han sido siempre idénticos tomando como validos los resultados de la figura (**Fig.4.7**).

```
root@mtx-1:~# iperf -c 192.168.2.2 -M 1000 -t 60
WARNING: attempt to set TCP maximum segment size to 1000, but got 536
------------------------------------------------------------
Client connecting to 192.168.2.2, TCP port 5001
TCP window size: 16.0 KByte (default)
------------------------------------------------------------
[  5] local 192.168.2.4 port 1028 connected with 192.168.2.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0-60.2 sec  26.4 MBytes  3.68 Mbits/sec
```

Fig.4.7: Cliente

Como se puede ver, el ancho de banda máximo obtenido es de 3.68Mbps en cada uno de los enlaces, con lo que el throughput máximo del escenario completo es de 7.36Mbps (la suma de ambos).

4.2.2 Pruebas con canales iguales.

Siguiendo el mismo escenario mostrado en **Fig.4.5**, modificaremos para la prueba únicamente los canales de las diferentes celdas configuradas anteriormente asignándoles el mismo canal para ambas. A continuación pasaremos a caracterizar el retardo y el throughput del mismo.

4.2.2.1. Caracterización del retardo.

Primeramente, al igual que en los anteriores apartados, se ha medido el retardo obtenido en ambos flujos simultáneamente. Tras realizar varias pruebas, el retardo medio obtenido para el flujo 1 (F1) ha sido de 2.606ms y el retardo obtenido en el F2 ha sido un poco superior rozando los 3ms.

Comparándolo con los resultados obtenidos en la prueba realizada usando canales diferentes no se aprecia ninguna diferencia aparente, esto es porque para medir el retraso los paquetes que se envían son muy pequeños siendo como consecuencia el tiempo de procesado también muy corto, con lo que la interferencia para medir los retardos no nos afecta demasiado.

4.2.2.2. Caracterización del ancho de banda.

Para la caracterización del rendimiento de la red, nuevamente se ha hecho uso de la herramienta IPERF, configurando para esta situación como nodos servidores los nodos 4 y 3, y como nodos clientes los nodos 1 y 2.

En esta prueba se ha podido comprobar claramente la importancia que tiene la asignación de canales, puesto que en este escenario la interferencia que se producía a la hora de generar un flujo provocaba la caída del otro encontrándonos con la situación de que en uno de los flujos se estaba trabajando con un ancho de banda alrededor de los 3.8 Mbps y en el otro flujo no se transmitía nada.

Por lo tanto el throughput total asignando a la red un único canal es alrededor de 3.8 Mbps, a diferencia de los 7.36 Mbps proporcionados por la misma red pero con canales diferentes.

4.3. Prueba 3: Escenario Mixto

Para esta última prueba, en el escenario que a continuación se muestra (**Fig.4.8**) nos vamos a encontrar con dos flujos de datos simultáneos hacia uno de los nodos, y se estudiará el comportamiento de la red en situaciones extremas.

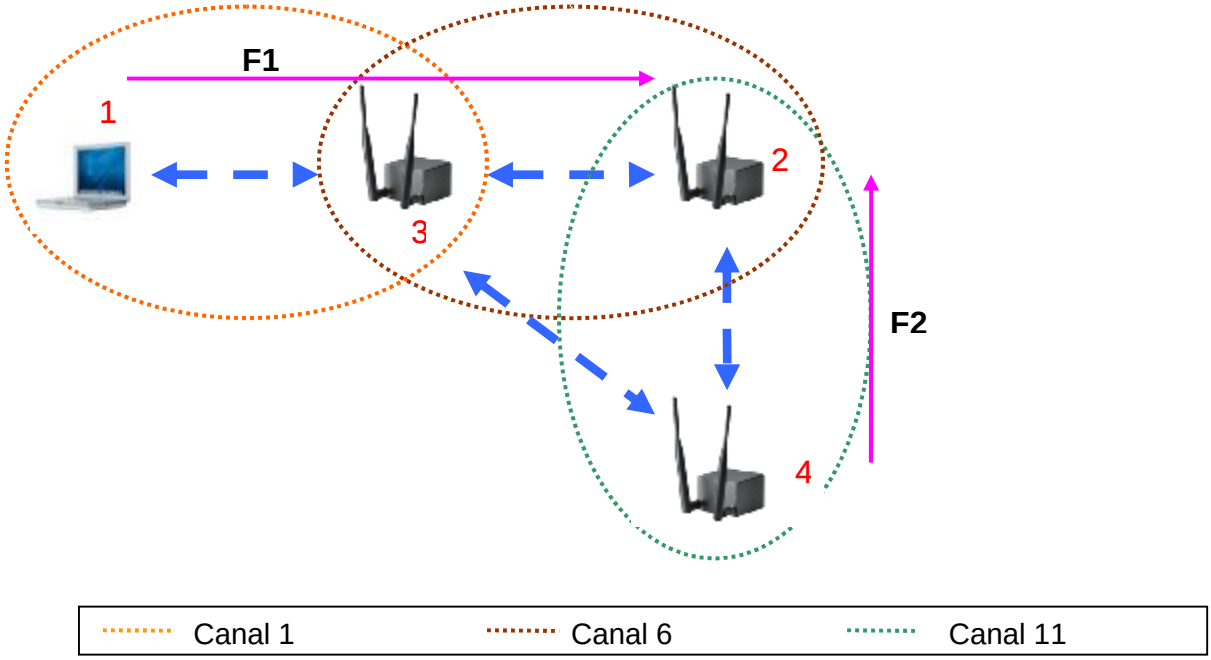

Fig.4.8: Escenario 3

Como se ha comentado al principio, en este escenario circularán dos flujos de datos a la vez. Estos flujos irán del nodo 1 al nodo 2 (**F1**) y del nodo 4 al nodo 2 (**F2**).

4.3.1. Pruebas con canales diferentes.

Para este escenario, los clusters serán dos, formados por los nodos 1-3 y 2-4 respectivamente, y para las comunicaciones intra-clusters se usará el canal 6, tal y como se puede observar en la figura (**Fig.4.8**).

Siguiendo la misma plantilla que se ha usado para realizar el procedimiento de la caracterización del escenario, primeramente se medirá el retardo para cada uno de los flujos, y a continuación se medirá el throughput total de la red.

4.3.1.1. Caracterización del retardo.

Nuevamente, haciendo uso de la herramienta PING, se ha caracterizado el retardo del escenario. Comentar como recordatorio que el resultado que nos proporciona esta herramienta es el tiempo de ida y de vuelta del paquete emitido.

Para este escenario, los resultados obtenidos para el flujo 1 (F1) ha sido de 5,072 ms y para el flujo 2 (F2) ha sido de 2,189ms. Como se puede observar en esta ocasión, los tiempos obtenidos son razonablemente coherentes, puesto que, partiendo de que en el lugar donde se realizan las pruebas se está libre de interferencias (el sótano de la universidad) y que los canales usados no se interfieren entre ellos, si se multiplica el retardo obtenido en el flujo 2 (1 salto) por el número de saltos que se realiza en el flujo 1 (2 saltos), se comprueba que el resultado obtenido es de 4,378 ms al cual si se le añade el tiempo que tardan los cubos en conmutar y procesar los paquetes, se nos va a los 5ms que nos ha dado en la medida.

4.3.1.2. Caracterización del ancho de banda.

A continuación, con la ayuda del IPERF, se ha pasado a realizar las medidas de rendimiento que proporciona la red. En esta ocasión se han realizado tres medidas, siendo muy similares entre ellas tal y como se muestra en la tabla (**Fig.4.9**).

Flujos	Pr1	Pr2	Pr3	Media
Flujo 1	2.14Mbps	1.82Mbps	1.92Mbps	1.96Mbps
Flujo 2	3.2Mbps	2.15Mbps	2.75Mbps	2.7Mbps

Fig.4.9: Resultados IPERF

Los resultados obtenidos no son al 100% los resultados que se esperaban, especialmente a lo que el flujo 1 se refiere, puesto que se esperaba que este flujo fuera igual que el flujo 2, aunque tras realizar diferentes consultas, se ha llegado a la conclusión de que esta imperfección viene producida por causa del Hardware o del Software usado por los cubos.

Así pues, el throughput total del presente escenario es de 4,66Mbps (la suma de los valores medios de los flujos).

4.3.2. Pruebas con canales iguales.

Por último se van a realizar para este escenario las pruebas de rendimiento para el caso de asignar en todos los enlaces un mismo canal común para todos, y comprobar finalmente como afecta la interferencia.

4.3.2.1 Caracterización del retardo.

Tal y como ocurrió en el apartado 4.2.2.1, en el que la diferencia entre calcular el retardo usando 1 canal o canales diferentes era inapreciable, se ha vuelto a dar el caso nuevamente en este escenario.

Tras realizar PINGS en los diferentes flujos, los datos obtenidos para los flujos 1 y 2 han sido de 5.236 ms y de 2.2 ms respectivamente. Siendo una diferencia inapreciable frente a los que se obtuvieron usando diferentes canales.

4.3.2.2. Caracterización del ancho de banda.

Finalmente para esta caracterización se volvió ha hacer uso de la herramienta IPERF. Para la realización de esta prueba, los nodos de los extremos se configuraron como servidores.

Tras realizar numerosas medidas, los datos que se obtuvieron de media en esta ocasión fueron: para el flujo 1 (F1) 135 Kbps, y para el flujo 2 (F2) 3 Mbps.

Un dato curioso que se pudo observar en esta prueba, fue comprobar que el flujo que siempre ha estado más afectado era el flujo 1, como consecuencia del número de señales interferentes que recoge el nodo 3. El nodo 3, como se puede observar en la figura (**Fig.4.8**), por estar en esa posición recibe dos señales interferentes, una procedente del nodo 2, y otra procedente del nodo 4. A diferencia de éste, los nodos 2 y 4 sólo reciben una señal interferente procedente del nodo 3. Es por esta causa, la caída importante que tiene el flujo 1, porque el número de señales interferentes al nodo 3 es superior al de los nodos que forman el flujo 2.

El thoughput total de este escenario sin asignar canales se sitúa alrededor de los 3,14 Mbps, siendo nuevamente un valor inferior al throughput obtenido anteriormente asignando diferentes canales (4,66 Mbps)

Capítulo 5: Conclusiones y líneas futuras

A lo largo de este proyecto se ha ido descubriendo la importancia que tiene el hacer un buen uso del espacio radioeléctrico del que se dispone para la creación de redes WLAN ad-hoc.

Habitualmente se lanzan nuevas tecnologías sin pensar que puede suceder en un futuro. Era evidente el día que se sacó al mercado la tecnología wi-fi que en un futuro no muy lejano, la gran mayoría de los usuarios iban a disponer de un dispositivos wireless en sus PC's ya sean en los que se disponen en oficinas o en sus hogares.

Como bien se sabe, las velocidades de transmisión son un punto muy importante para la realización de videoconferencias, ejecución de tareas a tiempo real en ordenadores remotos, video streaming, etc.… La tecnología wireless a diferencia de la tecnología por cable, tiene unas velocidades de transmisión bajas, a veces insuficientes para la realización de las tareas mencionadas.

Como se ha comentado a lo largo del proyecto una correcta asignación de los canales con el fin de minimizar interferencias facilita que la eficiencia del enlace alcance valores máximos, es por ello que se quiere destacar la importancia que tiene el hacer un uso correcto del espectro frecuencial. Como se ha podido observar en las pruebas realizadas, si todos los nodos trabajan con el mismo canal, se tiene un serio problema de interferencias, provocando colisiones en los paquetes transmitidos y obteniendo como resultado muy bajas velocidades de transmisión. Por el contrario, usando una buena distribución de canales al proporcionar dos interfaces radio a los nodos, las velocidades de transmisión se incrementan notablemente, pudiendo realizar transmisiones de velocidades más elevadas.

Tras realizar los estudios de los diferentes algoritmos comentados en el capítulo 1 y evaluar cuál de ellos iba a ser el más apropiado para nuestras necesidades, se pasó a implementar un nuevo algoritmo, tomando como referencia uno de ellos, pero realizando las modificaciones oportunas para superar algunas de sus limitaciones.

Mediante la programación en C de las diferentes funcionalidades mencionadas en apartados anteriores, se ha conseguido una aplicación que implementada en APs IEEE 802.11 con dos interfaces radio, permite la agrupación de éstos en clusters para conseguir un uso más eficiente del espectro y obtener así, tal y como se muestra en las pruebas de evaluación realizadas, una mejora en las prestaciones de la red.

Las principales características que ofrece el programa diseñado y las mejoras más destacables con respecto a las referencias estudiadas en el primer capítulo son:

- Se permite la movilidad y aparición y desaparición de nodos durante la fase de clustering: todo nodo que se conecte en ese periodo de tiempo, podrá participar en la elección del CH. Una vez se salga del clustering no se podrán incorporar más nodos hasta que se vuelva a entrar en la fase de clustering posteriormente.
- Todos los nodos tienen información en tiempo real de las modificaciones que se producen en cualquier nodo vecino a un salto, teniendo así las tablas siempre actualizadas.
- Puesto que los APs son dispositivos con memoria limitada, para el uso de las tablas o listas se trabaja con memorias dinámicas, haciendo un uso eficiente de los recursos de los mismos.
- Se trabaja a nivel 2, evitando el uso de cabeceras innecesarias, y transmitiendo únicamente los datos que son realmente de interés.
- El código utilizado es sencillo de entender, es rápido de procesar y eficaz para su finalidad.
- La asignación de un canal para el cluster por parte del CH no se realiza de manera unilateral sino que se trata de evitar las posibles interferencias procedentes de otros clusters al permitir el intercambio de cierta información entre los nodos y su CH.

Aún y así, el desarrollo futuro del algoritmo debería contemplar nuevas mejoras para solucionar problemas o limitaciones detectadas. Una de las implementaciones futuras que debería estudiarse seria cómo hacer que el algoritmo se ejecute cíclicamente. Hasta ahora, el programa implementado sólo se ejecuta una única vez, generando los clusters correspondientes y asignando canales a cada uno de ellos. Pero ahora bien, este tipo de redes esta pensada para soportar movilidad entre los diferentes nodos. En nuestro algoritmo faltaría por implementar la opción de mejorar el dinamismo para soportar la aparición de nuevos nodos y su desaparición, ya sea porque el nodo se haya desplazado y no alcance su cobertura, o que haya dejado de funcionar.

Por otra parte, como futuras mejoras del programa, en el apartado de asignación de canales, se ha implementado a la hora de escoger el canal óptimo para un cluster, que en el caso de que los tres canales de los que se disponen ya estén escogidos se cogiera por defecto el canal 1. Dejándolo de esta forma nos podemos encontrar que al asignar el canal 1 por defecto, se provoquen muchas interferencias entre los nodos frontera del cluster (que ya se habían asignado dicho canal anteriormente) y los nodos frontera del cluster que están asignando el canal. Una propuesta para mejorar esta situación, es la evaluación de las interferencias mediante medidas de carga y nivel de señal recibidos para cada uno de los canales disponibles, pudiendo así hacer uso de todo el espectro disponible y no sólo de los tres canales ya mencionados.

Debido a que la banda de frecuencias en la que se trabaja (2,4 GHz), proporciona un bajo número de canales, se pensó en la utilización del estándar

802.11a para las tareas de señalización y comunicación inter.-cluster. Con el hecho de utilizar este estándar lo que se evitó es eliminar un canal, pudiéndose utilizar de esta forma para la configuración de las antenas secundarias y reduciendo la probabilidad de encontrarse clusters vecinos con canales iguales. Por otra parte, el estándar 802.11a trabaja en la banda de 5 GHz, de esta manera no se provoca ningún tipo de interferencia en los canales usados en la banda de 2.4 GHz.

Ya entrando en un ámbito más comercial del producto, también se propone el cambiar las ID's que se le asignan a los nodos por las direcciones MAC de cada uno de los nodos, puesto que teóricamente estas direcciones no están repetidas. La ventaja que obtendríamos haciendo este cambio, es que guardaríamos las MAC's y a la hora de asignar un CH los mensajes en vez de enviarlos en broadcast, se mandarían de forma unicast, evitando el tener que poner filtros en el programa. Como "desventaja" (y lo pongo entre comillas porque no acabamos de llenar el campo mínimo de datos de una trama UDP), es el hecho de transmitir y almacenar un ID como el que se ha asignado en la configuración es que solo ocupa 1 Byte, en cambio una dirección MAC ocupa 6 Bytes. También se puede tener una tabla ID **à** MAC para seguir trabajando con IDs de 1Byte pero poder mandar mensajes unicast

Se puede decir que este proyecto no introduce contraindicaciones de carácter medioambiental ya que estamos utilizando tecnología inalámbrica que cumple con todas las normas de medio ambiente. La banda utilizada es un rango de frecuencias libres (2,4 GHz), por lo que cumple con la normativa IEEE 802.11 y con las leyes vigentes (Ley General de Telecomunicaciones). Además, la potencia a la que transmitimos para realizar las pruebas está dentro del margen de las normativas estatales y autonómicas.

6. Bibliografía

[1] Gerla, M.; Tsai, J.T.-C. (1995).”Multicluster, mobile, multimedia radio network”, ACM/Baltzer Journal of Wireless Networks. Vol 1 (Nº3), p. 255-265.

[2] Zhu, J.; Roy, S.; “802.11 Mesh Networks with Two-Radios Acces Points”

[3] Chunhung Richard Lin and Mario Gerla. (1997).” Adaptive Clustering for Mobile Wireless Networks”, IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, VOL. 15, NO. 7, pág: 1265-1275.

[4] Kaushik R. Chowdhury, Pritam Chanda, Dharma P. Agrawal, and Qing-An Zeng; “DCA – A Distributed Channel Allocation Scheme for Wireless Sensor Networks”

[5] Ruiz Lopez,D., Oliveras Pla,M. Redes mesh basadas en puntos de acceso inteligentes 802.11 Open Source (|||). Trabajo Final de Carrera, EPSC (UPC). Departamento Ingeniería Telemática, Febrero 2006.[Biblioteca Escola Politécnica Superior de Castelldefels]

[6] ThePacketFactory. ThePacketFactory. Disponible en: <http://www.packfactory.net>

[7] García, V. Redes mesh basadas en puntos de acceso inteligentes 802.11 open source (||). Trabajo Final de Carrera, EPSC (UPC). Departamento Ingeniería Telemática, Septiembre 2005. [Biblioteca Escola Politécnica Superior de Castelldefels]

[8] Meshcube.org. The meshing community webside. Disponible en: <http://www.meshcube.org>

[9] LinuxCommand.org.Linux Programmer’s Manual – IWCONFIG. Disponible en: <http://linuxcommand.org/man_pages/iwconfig.html>

[10] Debian.Debian – IPERF. Package :iperf(2.0.2-1, 1.7.0-1) Disponible en <http://packages.debian.org/unstable/net/iperf>

[11] Redes inalámbricas. Wi-fi, el futuro de la comunicación Introducción a las redes inalámbricas <http://www.mailxmail.com/curso/informatica/wifi/capitulo1.htm>

[12] Wireless Tools for Linux. Disponible en: <http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html>

[13] LINUX Tutorial pthreads. Disponible en: <http://members.tripod.com/webprototype/tutorial_pthreads_1.html>

[14] Ejemplos sencillos de programación avanzada en C en Linux. Disponible en: <http://www.abcdatos.com/tutoriales/tutorial/l6606.html>

[15] Tutoriales de Programación C. Disponible en: <http://www.abcdatos.com/tutoriales/programacion/c.html>

[16] La Web del Programador. Disponible en: <http://www.lawebdelprogramador.com/>

Escola Politècnica Superior de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

ANNEXOS

TÍTULO: Asignación de Canales en redes ad-hoc 802.11 multi-radio

TITULACIÓN: Ingeniería Técnica de Telecomunicaciones, especialidad Telemática

AUTOR: Manuel José Coronado Francisco

DIRECTOR: Eduard García Villegas

FECHA: 17 de Julio de 2006

A. Anexo A: Código del algoritmo

```
/*##############################################################################
##############################################################################
TFC/PFC - Asignación de Canales en redes ad-hoc 802.11 multi-radio
Manuel José Coronado Francisco
##############################################################################
##############################################################################
*/

#include <stdio.h>
#include <string.h>
#include "func.h"
#include <stdlib.h>
#include <time.h>
#include <pthread.h>

pthread_mutex_t mut;
unsigned char id, cid;
nodo *lista_vecinos;
char datos[5];
int pendiente = 0;
Tch_vecino *Tch_vecinos; // vector en el que se guardaran los id's de los ch de los clusters vecinos
int val=0;
int cont=0;
pthread_t e;

#define BUF_SIZE ETH_FRAME_TOTALLEN

extern char DEVICE[];

typedef struct nodo{
	unsigned char id;
	unsigned char id_ch;
}nodo;

typedef struct Tch_vecino{
		unsigned char id;
		unsigned char canal;
		unsigned char asignado;
}Tch_vecino;



char DEVICE[6];

void enviar_mensaje(char *datos);
int recibe_msg(unsigned char *m);
int guardar_nodos (unsigned char id1, unsigned char cid1, nodo *lista_vecinos, int *val);
void *algoritmo(void *q);
void catch_alarm(int sig_num);
void *envio(void *q);
void *reenvio_cid(void *q);
void catch_alarm1(int sig_num);
void *asignacion_canal(void *q);
void conf_ant2(unsigned char canal);
```

```
/* al inicializar el programa se le ha de dar el ID del nodo*/

int main(int argc, char *argv[]){


        unsigned char p='p';
        char tipo='t'; //valor provisional
        pthread_t t1;
        pthread_t reenvio;
        pthread_t canal;
        pthread_mutex_init (&mut, NULL);

        lista_vecinos =(nodo*)malloc(sizeof(nodo));

        Tch_vecinos = (Tch_vecino*)malloc(sizeof(Tch_vecino));

        if (argc != 2){
                perror("Indica la interficie");
                exit(1);
        }

    printf("El device es: %s \n", argv[1]);          //para el device
        strcpy(DEVICE,argv[1]);

    printf("Introduzca el ID del nodo\n");
    scanf("%c%c",&id, &cid);

        cid = id ;

    sprintf(datos,"%c%c%c%c",tipo,p,id,cid);

        enviar_mensaje(datos);

        pthread_create (&t1, NULL, algoritmo, NULL);      // Se ejecuta el algoritmo de Clustering
        pthread_join(t1,NULL);

        pthread_create (&reenvio, NULL, reenvio_cid, NULL);      // Se ejecuta el reenvio
        pthread_join(reenvio,NULL);

        pthread_create (&canal, NULL, asignacion_canal, NULL);   // Se ejecuta algoritmo de
                                                                 asignacion de canales
        pthread_join (canal,NULL);


}


void *algoritmo(void *q){          //Thread del algoritmo de Clustering

    int enviar = 0;             // cuando esta a 1 enviara mensaje
    int aux,x=0;
    unsigned char p='p';
    unsigned char r='r';
    char tipo='t';                  //valor provisional
    unsigned char mens[5];

        signal(SIGALRM,catch_alarm);

        alarm(10);
```

```
while(1){
        enviar = 0;

        if(recibe_msg(mens)==0 ){          // esperamos a la recepcion de un mensaje
                                           // si es =0 significara que es de nuestro protocolo
                if(mens[0] == 't' ){
                        if (mens[2] != id){

                                if (mens[1] == p)
                                    enviar = 1;

                                aux=guardar_nodos(mens[2], mens[3], lista_vecinos, &val);  //
guardamos/actuailzamos mensaje en lista_nodos

                                if(aux != -1){
                                        alarm(20);
                                        printf ("Tabla lista_vecinos\n\n");
                                    for (x=0;x<val;x++){
                                                printf("ID = %c --> CID = %c\n"
,lista_vecinos[x].id,lista_vecinos[x].id_ch);
                                    }


                                    if(lista_vecinos[aux].id == lista_vecinos[aux].id_ch){    //Es ch?
                                                if(cid>(lista_vecinos[aux].id)){              //tiene
                                                                                              un valor
                                                                                              menor?
                                                        pthread_mutex_lock(&mut);
                                                        cid = lista_vecinos[aux].id;
                //modificamos nuestro cid
                                                        pthread_mutex_unlock(&mut);
                                        enviar = 1;
                                         }
                                        }

                                        if((lista_vecinos[aux].id == cid)&&(lista_vecinos[aux].id !=
lista_vecinos[aux].id_ch)){
                                                        //Â nuestro ch, sigue siendo ch? si es que no
                                                        actualizamos
                                                pthread_mutex_lock(&mut);
                                                cid = id;
                                                //buscar mejor ch de la lista nodos
                                                for(x=0; x<= val; x++){ //se recorre TODA la lista,
                                                                        por si hay algun nodo con
                                                                        id menor
                                                        if(((lista_vecinos[x].id) ==
(lista_vecinos[x].id_ch))  && ((lista_vecinos[x].id)< cid)){
                                                                cid = lista_vecinos[x].id;
                                                        }
                                                }
                                                pthread_mutex_unlock(&mut);
                                                enviar = 1;
                                        }

                                        if(enviar == 1 && pendiente == 0){
                                                //Creacion thread de envio
                                                alarm(10);
                                                pthread_create (&e,NULL,envio,NULL);
```

```
                                        }
                                    printf(" el valor del id =%c\n el valor del cid=%c\n",id,cid);
                                }
                            }
                        }
                        }
        }

}


void *reenvio_cid(void *q){                 // Thread de reenvio de mensajes de informacion

        int al, x;
    unsigned char mens[5];

        if (cid == id){                     //soy CH

                signal(SIGALRM, catch_alarm1);

                alarm(20);

                //inicializo la alarma (t=20seg)

                while(1){
                        if(recibe_msg(mens)== 0){
                                if ((mens[0]=='i') && (mens[1]==id)){ // el mensaje es de info y es
                                                                        para mi???
                                        for (x=0;x<cont;x++){
                                                if (Tch_vecinos[x].id == mens[2]){
                                                        al=-1;
                                                        break;
                                                }
                                        }
                                        if (al != -1){
                                                Tch_vecinos[cont].id = mens[2];
                                                printf("Tabla CH vecinos = %c\n"
,Tch_vecinos[cont].id);
                                                cont ++;
                                        Tch_vecinos = (Tch_vecino*)realloc (Tch_vecinos,(cont+1)
* sizeof(Tch_vecino));
                                        }
                                }
                        }
                }

        }

        else {                                      //no soy CH
                sleep (rand()%10);
                printf("Tabla de vecinos que tenemos\n\n");
                for (x=0;x<val;x++){

                printf("ID = %c --> CID = %c\n",lista_vecinos[x].id, lista_vecinos[x].id_ch);

                        if (cid != lista_vecinos[x].id_ch){
                                sprintf(datos,"i%c%c",cid,lista_vecinos[x].id_ch);
                                printf("Enviamos a nuestro CH la trama %s",datos);
                                enviar_mensaje(datos);
```

```
                        }
                }
        }
}

void *asignacion_canal(void *q){

        int x, y, aux=0;
        unsigned char mens[5];
        unsigned char canal_usado;
        unsigned char envia[5];
        Tch_vecino a;

        if (id == cid){
                printf("cont=%d\n",cont);
                if (cont>0){        //ordenamos tabla CH vecinos
                        for(x=0; x<(cont-1); x++){
                                if (Tch_vecinos[x].id > Tch_vecinos[x+1].id){
                                        a= Tch_vecinos[x];
                                        Tch_vecinos[x]=Tch_vecinos[x+1];
                                        Tch_vecinos[x] = a ;
                                }
                        }
                }
                for (x=0; x<cont; x++){
                        if (Tch_vecinos[x].id < cid){
                                if (Tch_vecinos[x].asignado == 0)
                                        aux++;
                        }
                }
                if (aux == 0){
                        //asigno canal
                        printf("Tengo el ID mas bajo y asigno canal\n");
                        canal_usado = '1';
                        conf_ant2(canal_usado);

                }else while (aux != 0){
                        printf("NO Tengo el ID mas bajo y me espero\n");
                        if (recibe_msg(mens) == 0){         //esperamos que los Clusters con id_ch
                                                            menor se asignen un canal
                                if (mens[0]=='c'){
                                        for(x=0; x<cont ; x++){    // Guardamos el canal que han
                                                                   asignado y marcamos el campo
                                                                   asignado a 1
                                                if(Tch_vecinos[x].id == mens [1]){
                                                        Tch_vecinos[x].canal = mens[2];
                                                        Tch_vecinos[x].asignado = '1';
                                                        break;
                                                }
                                        }
                                        aux = 0; //Comprovamos si no hay ningun otro CH mas
                                                 prioritario para asignar canal
                                        for (x=0; x<cont; x++){
                                                if (Tch_vecinos[x].id < cid){
                                                        if (Tch_vecinos[x].asignado == '0')
                                                                aux++;
                                                }
                                        }
                                        if (aux == 0){
```

```
                                                printf("asigno ahora el canal, trama rec=
%s\n",mens);
                                                x=0;      //asigno canal
                                                canal_usado = '1';
                                                while (Tch_vecinos[x].asignado == '1'){
                                                        if(canal_usado == Tch_vecinos[x].canal)
                                                                canal_usado= canal_usado + 1;
                                                        if (canal_usado > '3')
                                                                canal_usado= '1';
                                                        x++;
                                                }
                                                conf_ant2(canal_usado);
                                        }
                                }
                        }
                }
                sprintf(envia,"c%c%c",cid,canal_usado);
                enviar_mensaje(envia);
        }
        if(cid != id){
                printf("soy gw\n");

                aux=0;
                while (aux == 0){
                        if (recibe_msg(mens) == 0){
                                printf("el mensaje ha llegado y es: %s",mens);
                                if((mens[0]=='c')&&(mens[1]==cid)){        //si el mensaje ha sido
                                                                        generado por nuestro ID_CH, configuramos
                                                                        las antenas
                                        aux=1;
                                        conf_ant2(mens[2]);
                                }
                                else{     //mientras estamos a la espera de que nuestro CH nos envie el
                                        canal para el cluster, vamos reenviando las tramas que
                                        vallamos cogiendo
                                        if (mens[0]=='c'){
                                                sprintf(envia,"c%c%c",mens[1],mens[2]);
                                                enviar_mensaje(envia);
                                        }
                                }
                        }
                }

                sprintf(envia,"c%c%c",cid,mens[2]);
                enviar_mensaje(envia);

        }
}



void catch_alarm(int sig_num){
        alarm(0);
        printf("****Salimos del Clustering****\n");
        if (pendiente == 1)
                pthread_join(e, NULL);
        pthread_exit(NULL);
}
```

```
void catch_alarm1(int sig_num){
	alarm(0);
	printf("****Salimos de Reenvio****\n");
	pthread_exit(NULL);
}

void *envio(void *q){
	unsigned char envia[5];
	pendiente = 1;
	sleep(rand()%6);
	pthread_mutex_lock(&mut);
	sprintf(envia,"tr%c%c",id,cid);
	pthread_mutex_unlock(&mut);
	enviar_mensaje(envia);
	pendiente = 0;
}

void conf_ant2(unsigned char canal){

	switch (canal){
		case '1':  //system("iwconfig wlan1 essid Manu mode ad-hoc channel 1");
				printf("\nAntena secundaria configurada con el canal 1\n");
				break;

		case '2':  //system("iwconfig wlan1 essid Manu mode ad-hoc channel 6");
				printf("\nAntena secundaria configurada con el canal 6\n");
				break;

		case '3':  //system("iwconfig wlan1 essid Manu mode ad-hoc channel 11");
				printf("\nAntena secundaria configurada con el canal 11\n");
				break;
	}
}



/**
 * Procesa el mensaje recibido en el string data1
 * @param data1 puntero al vector tipo char
 */
void enviar_mensaje(char *datos){

	int s = 0;
	void* buffer = NULL;
	buffer = (void*)malloc(ETH_FRAME_LEN);		/*Buffer de la trama ethernet */
	unsigned char* etherhead = buffer; /*Puntero de la cabezera ethenet */
	unsigned char* data = buffer + 14; /*Userdata in ethernet frame*/
	struct ethhdr *eh = (struct ethhdr *)etherhead; /*Another pointer to ethernet header*/

	unsigned char src_mac[6];		/*our MAC address*/
	unsigned char dest_mac[6] = {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};//{0x00, 0x10, 0x5A,
0xDE, 0xA7, 0x2C};	/*other host MAC address, hardcoded..... :-(*/

	struct ifreq ifr;
	struct sockaddr_ll socket_address;
	int ifindex = 0;				/*Ethernet Interface index*/
	int i;
	int sent;					/*length of sent packet*/

	/*open socket*/
```

```
        s = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL));
        if (s == -1) {
                perror("socket():");
            exit(1);
        }

        /*retrieve ethernet interface index*/
        strncpy(ifr.ifr_name, DEVICE, IFNAMSIZ);
        if (ioctl(s, SIOCGIFINDEX, &ifr) == -1) {
                perror("SIOCGIFINDEX");
                exit(1);
        }
        ifindex = ifr.ifr_ifindex;

        /*retrieve corresponding MAC*/
        if (ioctl(s, SIOCGIFHWADDR, &ifr) == -1) {
                perror("SIOCGIFINDEX");
                exit(1);
        }
    for (i = 0; i < 6; i++) {
                src_mac[i] = ifr.ifr_hwaddr.sa_data[i];
        }

        /*prepare sockaddr_ll*/
        socket_address.sll_family   = PF_PACKET;
        socket_address.sll_protocol = htons(ETH_P_IP);
        socket_address.sll_ifindex  = ifindex;
        socket_address.sll_hatype   = ARPHRD_ETHER;
        socket_address.sll_pkttype  = PACKET_OTHERHOST;
        socket_address.sll_halen    = ETH_ALEN;
        socket_address.sll_addr[0]  = dest_mac[0];
        socket_address.sll_addr[1]  = dest_mac[1];
    socket_address.sll_addr[2]  = dest_mac[2];
    socket_address.sll_addr[3]  = dest_mac[3];
    socket_address.sll_addr[4]  = dest_mac[4];
    socket_address.sll_addr[5]  = dest_mac[5];
        socket_address.sll_addr[6]  = 0x00;
        socket_address.sll_addr[7]  = 0x00;


        /*prepare buffer*/
        memcpy((void*)buffer, (void*)dest_mac, ETH_MAC_LEN);
        memcpy((void*)(buffer+ETH_MAC_LEN), (void*)src_mac, ETH_MAC_LEN);
        eh->h_proto = ETH_P_NULL;

        strcpy(data,datos);

        /*send packet*/

        sent = sendto(s,buffer, 60/*ETH_FRAME_LEN*/, 0, (struct sockaddr*)&socket_address,
sizeof(socket_address));
        if (sent == -1) {
                perror("sendto():");
                exit(1);
        }
//      write(1,data,strlen(data));

        close(s);

    free(buffer);
```

```
}

/**
 *Procesa la recepcion de un mensaje
 *@param se le pasa una estructura tipo msg para que nos incluya los resultados
 *@result los resultados serÃ¡n:nÂª de campos y los datos del mensa, ademas nos devolverÃ¡:
 *          1 si no es un paquete de nuestro protocolo
 *          0 si la trama pertenence a nuestro protocolo
*/
int recibe_msg(unsigned char *m){

          int s = 0;
          void* buffer = NULL;
          buffer = (void*)malloc(ETH_FRAME_LEN);                    /*Buffer for Ethernet Frame*/
          unsigned char* etherhead = buffer;              /*Pointer to Ethenet Header*/
          struct ethhdr *eh = (struct ethhdr *)etherhead; /*Another pointer to ethernet header*/
          unsigned char* data = buffer + 14;

          unsigned char src_mac[6];             /*our MAC address*/

          struct ifreq ifr;
          struct sockaddr_ll socket_address;
          int ifindex = 0;                      /*Ethernet Interface index*/
          int i;
          int length=0;                                 /*length of received packet*/

          /*open socket*/
          s = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL));
          if (s == -1) {
                    perror("socket():");
               exit(1);
          }

          /*retrieve ethernet interface index*/
          strncpy(ifr.ifr_name, DEVICE, IFNAMSIZ);
          if (ioctl(s, SIOCGIFINDEX, &ifr) == -1) {
                    perror("SIOCGIFINDEX");
                    exit(1);
          }
          ifindex = ifr.ifr_ifindex;

          /*retrieve corresponding MAC*/
          if (ioctl(s, SIOCGIFHWADDR, &ifr) == -1) {
                    perror("SIOCGIFINDEX");
                    exit(1);
          }
        for (i = 0; i < 6; i++) {
                    src_mac[i] = ifr.ifr_hwaddr.sa_data[i];
          }

          /*prepare sockaddr_ll*/
          socket_address.sll_family   = PF_PACKET;
          socket_address.sll_protocol = htons(ETH_P_IP);
          socket_address.sll_ifindex  = ifindex;
          socket_address.sll_hatype   = ARPHRD_ETHER;
          socket_address.sll_pkttype  = PACKET_OTHERHOST;
          socket_address.sll_halen    = ETH_ALEN;
          socket_address.sll_addr[6]  = 0x00;
          socket_address.sll_addr[7]  = 0x00;
```

```
	/*Wait for incoming packet...*/
	while(1){
	length = recvfrom(s, buffer, ETH_FRAME_LEN, 0, NULL, NULL);
	if (length == -1) {
		perror("recvfrom():");
		exit(1);
	}
	/* Miramos los campos de la trama (Ethertype == 0x0 nuestro protocolo)*/
	if (data[0]=='t' || data[0]=='i' || data[0]=='c'){
		if( data[0]== 't' && data[2] == '1'){ // --> usado para realizar pruevas
			close(s);
			free(buffer);
			return 1;
		}

		strcpy(m,data);

		printf("Datos recibidos --> %s \n",m);
		close(s);
	free(buffer);
		return 0;
	} }

	close(s);

    free(buffer);
	return 1;
}

/**
 * Guarda o actualiza los nodos en la lista_vecinos
 * @param	unsigned char id es el id del nodo que manda la trama
 *		unsigned char cid es el id_ch del nodo que ha mandado la trama
 *		nodo *lista_vecinos es un vector dinÃ¡mico donde se guardaran los id's y los cid's de
todos los nodos vecinos
 *		int *val es un puntero que nos indicarÃ¡ cual es la Ãºltima posicion del vector
 * @result devuelve la posicion de memoria en la que ha almacenado el nodo entrante, la lista_vecinos
actualizada y el puntero de la misma incrementado
 */

int guardar_nodos (unsigned char id1, unsigned char cid1, nodo *lista_vecinos, int *val){

    int x;
    int find=0;

	for(x=0; x<= *val; x++){
	 if (lista_vecinos[x].id ==id1){
		find=1;
			break; //fuerza salida del bucle, y en x tenemos la posicion de la variable
	 }
    }

    if (find == 0){ // no se ha encontrado

		x = *val;

	 lista_vecinos = (nodo *) realloc(lista_vecinos,(x+1)*sizeof(nodo));
	 lista_vecinos[x].id = id1;
```

```
            lista_vecinos[x].id_ch =cid1;
            *val= x+1;
           }else{
                    if(lista_vecinos[x].id == id1 && lista_vecinos[x].id_ch == cid1)
                            return -1;
            lista_vecinos[x].id = id1;
            lista_vecinos[x].id_ch =cid1;
      }
      return (x);
}
```